September 1984

EVALUATION OF AUTOMATED DECISIONMAKING METHODOLOGIES AND DEVELOPMENT OF AN INTEGRATED ROBOTIC SYSTEM SIMULATION

Prepared by:

Dennis C. Haley
Bonni J. Almand
Mark M. Thomas
Linda D. Krauze
Keith D. Gremban
James C. Sanborn
Joy H. Kelly
Thomas M. Depkovich

This work was performed for NASA Langley Research Center under Contract NAS1-16759.

MARTIN MARIETTA AEROSPACE
DENVER AEROSPACE
P.O. Box 179
Denver, Colorado 80201

CONTENTS

Page

SUMMARY

Work tasks that require extensive manual labor in hazardous environments, (e.g., NASA advanced missions) would derive great benefit from increased use of automation technology. This report documents a NASA-sponsored activity to develop the specific technologies needed for increasing the role of automation in such missions and to implement a generic computer simulation capability for manipulator systems.

The computer simulation developed provides the ability to perform kinematic and dynamic analysis of user-defined manipulators. The user establishes a manipulator system model, including arms, load objects and an environment, in response to program prompts. The task profile is also specified interactively by the user and consists of motion segments utilizing position or rate control in joint or end-effector motion coordinates, and such nonmotion steps as GRASP (for grasping a load object).

The kinematic analysis provides positions, velocities, and accelerations of all parts of the system for the prescribed motion; it incorporates a robust iterative joint solution algorithm for generality. The dynamic analysis procedures include requirements analysis, which calculates the system loads for specific motions, and response simulation, which gives the motion trajectory resulting from a prescribed set of driving torques or feedback control law.

The analysis results can be displayed as printed tabular output or plots of the trajectories of relevant parameters. Animated graphic display is available during system creation and analysis to verify the system configuration and motion.

The specific automation technologies investigated include control system design, trajectory planning and image processing. A general overview of control system design is presented, with special emphasis on adaptive control and hybrid control of forces and positions. Computer simulation of these control modes is available.

Three algorithms for generating collision-free paths through cluttered environments were implemented and compared. The algorithms include a joint space search method, a method in which free space is represented by a collection of generalized cones, and an incremental motion procedure in which constraints are translated locally into joint coordinates. The benefits and drawbacks of each method are discussed.

The literature on edge detection is reviewed in this report. An edge-detection algorithm was implemented, along with a computer simulation of a manipulator-mounted camera for vision sensing.

Appendices to this report describe in detail the programming and use of the computer simulation package (available through COSMIC).

INTRODUCTION

Background

This document reports the results of work performed in Tasks 9 through 20 of contract NAS1-16759, Evaluation of Automated Decisionmaking Methodologies and Development of an Integrated Robotic System Simulation. It was prepared by Martin Marietta Denver Aerospace for the Langley Research Center of the National Aeronautics and Space Administration (NASA-LRC) in accordance with the contract statement of work. These tasks constitute Phases II and III of a multiphase activity addressing the technologies relevant to design and operation of advanced manipulator systems. Phase I of this activity concentrated on the identification and evaluation of artificial intelligence techniques applicable to NASA advanced missions and on developing a framework and mathematical models for the computer simulation of manipulator systems. The results were documented in 1982 (NASA Contractor Reports 165975, 165976 and 165977).

This project is motivated by the realization that NASA advanced missions require increasing use of automation technology for both economical and performance reasons. Factors influencing this trend include:

1) The cost of supporting man in the hostile space environment is much greater than that of supporting an unmanned system;

2) Human strength, dexterity, reach and precision are too limited for some applications;

3) A well-designed automated system provides more optimal control than a human controller;

4) Many mission tasks are highly repetitive and mundane. Automation technology is ideally suited for such applications, while humans tend to become bored and make more mistakes;

5) Automation systems are better suited for round-the-clock operations that more fully utilize limited resources on space missions.

Despite these drawbacks involved in direct human operation or control, the adaptability, resourcefulness and problem-solving ability of man are still needed for the complex dynamic environment associated with space applications. Current robot systems are limited to relatively simple, preprogrammed tasks in structured environments and incorporate very little machine intelligence and external sensing. To achieve the ambitious goals of the space program in such areas as space station and long-life reserviceable spacecraft, it is essential to reduce direct human control of the robotic systems. This reduction can most naturally occur over a four-phase development process.

The first phase is to develop the required system with man in the loop to provide control and the problem-solving functions. The second phase of robotic system evolution is to extract the man from the primary control loop to assume a supervisory role. In this role, the operator will perform the function of planning out a sequence of tasks to achieve a specific goal. The robotic system will perform the tasks of trajectory planning, obstacle avoidance, and joint control. In the third phase, the individual will be extracted one more level. In this phase, the operator will perform the function of establishing intermediate goals for the robotic system. The robotic system will perform the functions associated with breaking down the specific goals into individual tasks to be performed. The final phase of robotic system development will result in a fully autonomous robotic system.

To accomplish these goals requires dramatic improvement in some of the manipulator component technologies, especially in the fields of sensing, control and artificial intelligence. This contract activity addressed several of these issues; implementations were developed for image processing (edge detection), intelligent path planning, and advanced control strategies (including adaptive control and hybrid force/position control).

Development of the complex technologies associated with advanced automation in a timely and cost-efficient manner requires extensive use of computer simulation tools that allow implementations to be evaluated and compared before building hardware prototypes. The capabilities that a kinematic simulation (i.e., one in which motions, not forces, are considered) can provide include:

1) Find and display manipulator dexterity and workspace. This can be used to evaluate kinematic designs, suggest workcell arrangement (where feasible), and help design systems for maintenance and repair by automation;

2) Verify and implement path planning, including obstacle avoidance and singularity detection;

3) Evaluate improvement in system operation from some types of sensors such as proximity sensors or moving video cameras;

4) Determine potential speed of operation;

5) Training for teleoperator control using the simulation instead of hardware, and evaluate the different levels of human interaction in the control loop.

A dynamic simulation also includes the system forces and provides additional capabilities such as:

1) Verification and evaluation of controller designs, especially those that incorporate advanced control concepts. For example, adaptive control schemes often involve identification of system parameters from response information. With a dynamic simulation, the actual values for the parameters are specified so the identification scheme can readily be verified;

2) Verification of system performance, structural integrity, load distribution and component stress levels;

3) Testing the use of force-related sensors such as a force/torque wrist or gripper force sensors in system operation;

4) More accurate simulation for teleoperator training, including control involving force-feedback or dynamic interactions with the environment.

A dynamic simulation package for the entire manipulator system forms an indispensable tool for design and development of automation implementations for NASA advanced missions.

Contract Objectives

The primary objectives of this contract activity are the implementation of an integrated robotic simulation package and development of the technologies relevant to operation of advanced manipulator systems. The research performed and software developed during the contract phases reported here focused on the following capabilities:

1) Computer simulation of a robot in operation, including system kinematics and dynamics, interactive control of program execution by the user, and graphic display of the system operation;

2) Computer simulation of multisensor grippers;

3) Control concepts for manipulators incorporating adaptive techniques and control of force levels as well as positions;

4) Trajectory planning for manipulator motions in unstructured environments;

5) Image processing and simulation of a manipulator-mounted video camera.

Report Organization

This report consists of a main text that describes the study results, including the technical aspects of the robotic simulation (ROBSIM) package and of the automation technologies investigated, along with two appendices that document the computer implementation of ROBSIM. ROBSIM consists of three functional packages (Fig. 1):

1) System definition function - Allows the user to interactively create a manipulator system containing manipulators, load objects and an environment. This package addresses Task 12 of the contract;

2) Analysis tools function - Contains requirements analysis and response simulation options for investigating the load/motion relationships of a manipulator system in operation, and addresses Tasks 14, 15 and 16 of the contract;

3) Postprocessing function - Aids in interpreting the analysis results by graphic display of results, replay of simulated motion, etc.

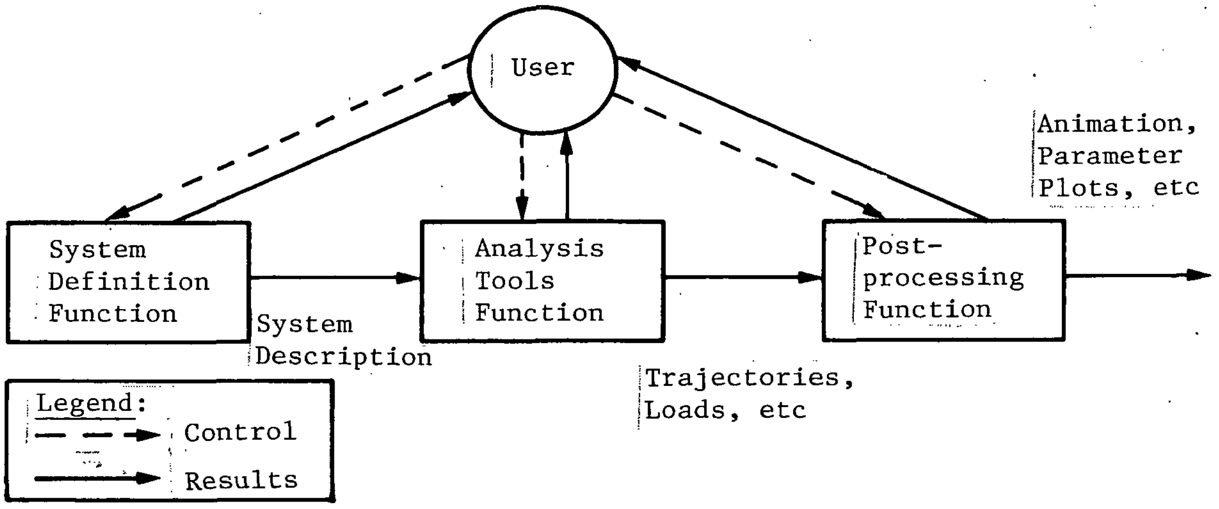


Figure 1.- ROBSIM functional blocks.

The following three sections of this document describe the technical approach employed in each of these functions.

The next section after these sections discusses advanced controller concepts and their evaluation and simulation. It covers work performed in Tasks 13, 17 and 19 of the contract. The section after this describes trajectory planning research performed for Tasks 9 and 18. Three methods for obstacle avoidance were implemented and evaluated. Video simulation and image processing are discussed in the following section. The video simulation module and edge detection algorithm address the requirements of Tasks 10 and 11. The work in Task 20, hardware vs software validation, forms the next section. This section was addredded with cooperation from NASA-LRC. Validation efforts were carried out on a 2 DOF planar arm at Martin Marietta and a 6 axis PUMA robot at NASA-LRC. The final section of this text summarizes the results of this activity and describes avenues for further efforts to expand, enhance and utilize capabilities developed during the performance of this contract.

This document has two appendices available through COSMIC.* Appendix A is a ROBSIM User's Guide and describes the steps involved in running the program. Appendix B provides the programmer with additional information concerning program implementation. This appendix and the in-code documentation provide sufficient information to allow modification of the program for special-purpose applications.

* Inquiries concerning the program ROBSIM, Appendix A (User's Guide), and Appendix B (Programmer's Guide) should be directed to: COSMIC, 112 Barrow Hall, University of Georgia, Athens, GA 30601.

MANIPULATOR SYSTEM DEFINITION

This section describes the methods implemented in the ROBSIM package for defining *a robotic manipulator system. The discussion of the system definition* is separated into four subsections:

1) Manipulator arm creation/modification;

2) Environment creation/modification;

3) Load objects creation/modification;

4) System creation.

A system is actually composed of one or more of the components listed in items 1) through 3) above.

Manipulator arm creation/modification is used to define the mass and geometric properties of one robot arm. This includes properties for the base, all joint/link pairs, and the end-effector (also called "tool" or "hand"). Detailed geometries may also be defined for each part of the arm. These, however, are used for the graphics displays that accompany the ROBSIM framework and do not affect arm motions or any of the analyses methods described later.

Environment creation/modification is used to simulate immovable objects in the workspace of the manipulator arm. Currently the usefulness of the environment definition is limited to graphic displays and does not affect or hinder the manipulator motion in any way.

Creation/modification of load objects is very similar to that of the environment, with the exception that load objects may be moved around the workspace by one or more manipulator arms.

System creation is used to bring together the components into one coherent group. A system may contain as little as one manipulator arm or multiple arms, a detailed environment, and a group of load objects. Each component is placed in the system with respect to a reference or world coordinate system. Table 1 lists the notations used in this section.

TABLE I. - SYSTEM DEFINITION SYMBOLS

X_i	local coordinate system for component i
$[P_i]$	coordinate system i-to-world coordinate system transformation matrix
$[{}_jP_i]$	coordinate system i to j transformation matrix
$\underline{R}_i$	vector defining location of origin of local coordinate system of component i in world coordinate system
$\underline{r}_i$	vector defining the location of i in world coordinates
${}_j\underline{h}_{ij}$	vector from the origin of joint i to origin of joint j in the i coordinate system
${}_j\underline{h}_{jj}$	vector from joint j origin to link j center of gravity in the j coordinate system
θ_j	angular displacement of joint j
$[I_i]$	inertia matrix of component i
m_i	mass of component i
b_i	radius of simple cylinder representation of component i
l_i	length of simple cylinder representation of component i
$E1_i$, $E2_i$	endpoints of simple cylinder representation of component i
Note: If i or j is a b, this denotes the manipulator base; if an L it refers to a load object.	

Arm Creation/Modification

The method of manipulator arm definition implemented in ROBSIM separates components of the arm into one of three categories:

1) Base;

2) Joint/link pairs;

3) End-effector.

The base of the arm is defined first, followed by each of the joint/link pairs, and finally the end-effector. Each component of the arm is defined within its own local coordinate system, which is referenced back to the world coordinate system.

Base. - The local coordinate system of a manipulator base is denoted X_b. The location and orientation of this local coordinate system with respect to the world system is defined by the user. $\underline{R}_b$ is the vector containing the world system coordinates of the origin of the X_b coordinate system. Orientation of X_b with respect to the world system is given by the matrix $[P_b]$. The columns of $[P_b]$ contain direction cosines of each axis of X_b with the world coordinate system axes. This orientation transformation can be used to locate any point of interest with respect to the world using the relation

$$\underline{r}_i = \underline{R}_b + [P_b]\,{}_b\underline{r}_i$$

where ${}_b\underline{r}_i$ is the location of the point of interest in the base coordinate system. The base itself is modeled as a simple cylinder with the centerline along the base coordinate system x-axis.

In the current version of ROBSIM, the base is stationary once placed in a system. For this reason, mass properties are not defined, only geometric properties are.

Joint/link. - The majority of the arm is described as joint/link pairs. Each joint is defined to have a specific location and orientation. However, all mass properties of the joint/link pair are associated with the link. The local coordinate system for each joint is denoted by X_i where i is the joint number. The joints of the arm are numbered consecutively, starting with joint 1 next to the base. Three types of joints are available for ROBSIM modeling:

1) Hinge joints;

2) Swivel joints;

3) Sliding joints.

Hinge joints rotate about the joint local y-axis, swivel joints about the x-axis, and sliding joints translate along the x-axis. Figure 2 shows each of the joint types. The location of each joint j is specified by the vector ${}_i\underline{h}_{ij}$, the vector from the previous joint i (or base if j = 1) to the current joint j in the X_i coordinate system. To locate this joint in the world coordinate system, the transformation matrix $[P_i]$ and local joint location vector ${}_i\underline{h}_{ij}$ are multiplied and added to the world system location vector of joint i

$$\underline{R}_j = \underline{R}_i + [P_i]\,{}_i\underline{h}_{ij}$$

or

$$\underline{R}_j = \underline{R}_i + [P_i]\,{}_i\underline{h}_{ij} + [P_j]\begin{pmatrix}\theta_j \\ 0 \\ 0\end{pmatrix}$$

for sliding joints, where θ_j is the linear displacement for joint j and $[P_j]$ is the transformation matrix from the joint j coordinate system to the world coordinate system.

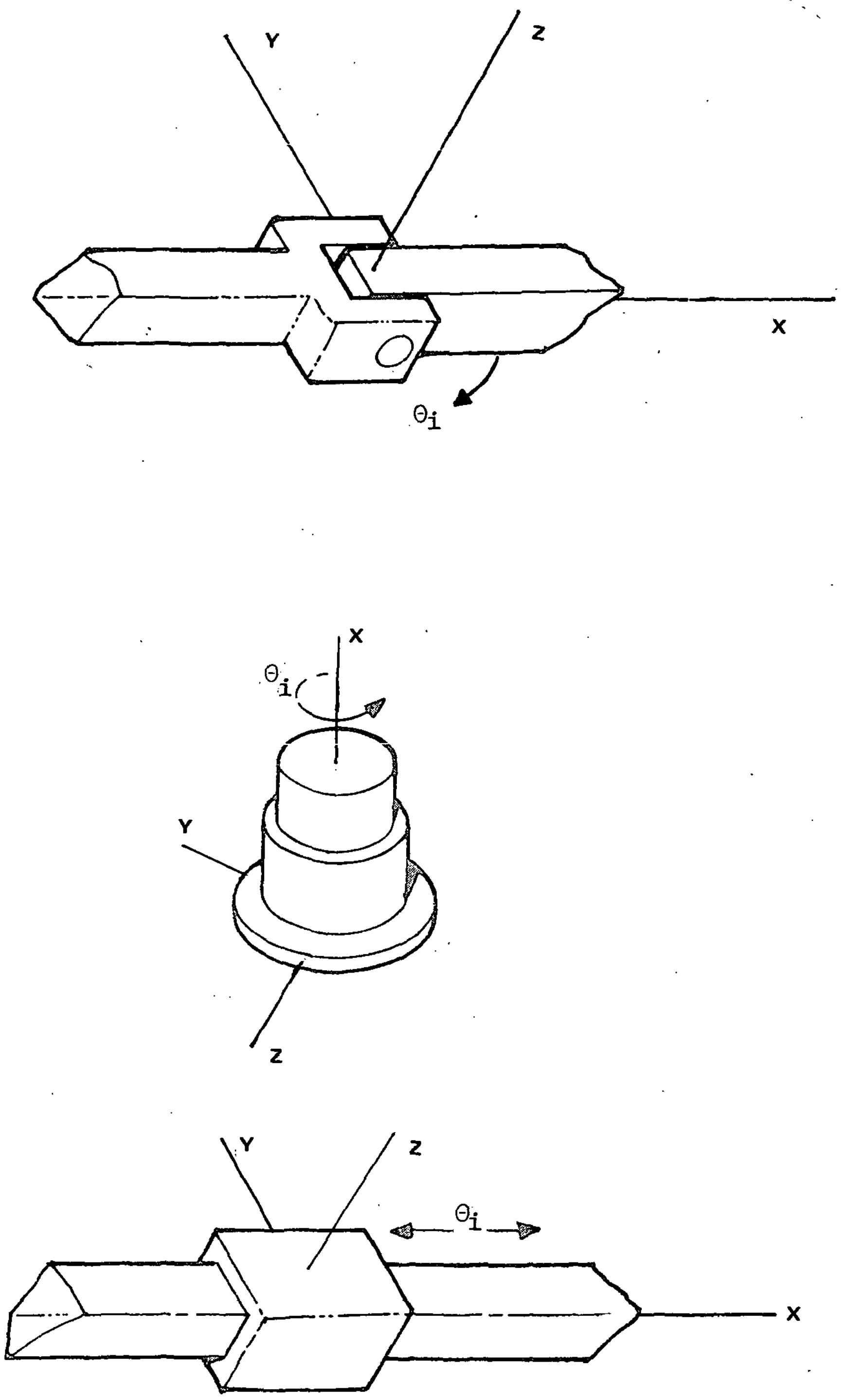


Figure 2.- Hinge, swivel and sliding joints.

Joint orientation is also defined with respect to the previous joint with the transformation matrix $[{}_iP_j]$. Each column contains the direction cosines of an axis of the X_j coordinate system to the X_i coordinate system axes. The transformation matrix $[P_j]$ is then calculated from

$$[P_j] = [P_i][{}_iP_j]$$

The capability for user-defined effective actuator parameters is included for each joint. These parameters are:

1) Actuator torque constant;

2) Motor gear ratio;

3) Actuator amplifier gain;

4) Back EMF constant;

5) Motor effective inertia;

6) Motor winding resistance;

7) Motor winding inductance;

8) Coulomb friction coefficient;

9) Static friction coefficient;

10) Effective viscous damping.

The use of these parameters is discussed in detail in the Analysis Capability section. The initial joint position $\hat{\theta}_j$ is the displacement of the joint measured from its original location and orientation.

The link accompanying each joint is defined as being placed after the joint. For example, link 2 goes from joint 2 to joint 3. The link is defined initially as a simple cylinder with its centerline along the joint local x-axis. Endpoints of the link are just coordinates along this axis. Figure 3 shows joint/link sequencing and locations. ROBSIM puts the center of mass at the geometric center of the link. The link center of mass vector ${}_j\underline{h}_{jj}$ then has the coordinates.

$${}_j\underline{h}_{jj} = \left(\frac{E2_i - E1_i}{2} , 0, 0 \right)$$

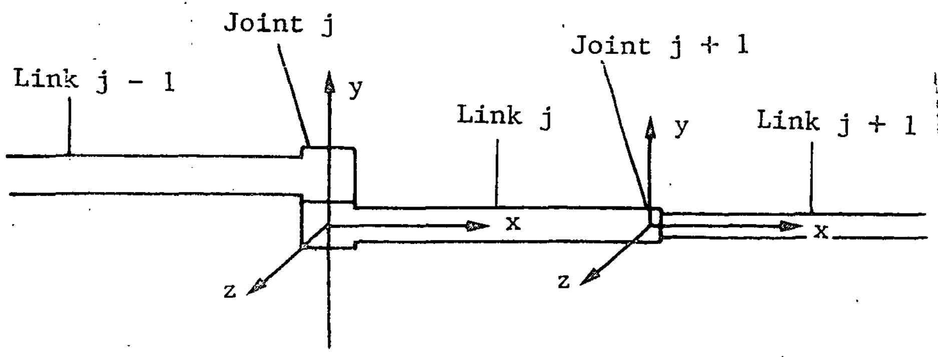


Figure 3.- Joint/link sequencing.

Multiplying by $[P_j]$ transforms this vector to the world coordinate system. As an alternative to this placement of the center of mass, an arbitrary center of mass may be defined by user input of the local system coordinates of the desired center of mass. The algorithm implemented in ROBSIM for calculating the link inertia matrix uses the following equations for computing the diagonal terms

$$I_{i_{1,1}} = .5\, m_i b_i^2$$

$$I_{i_{2,2}} = \frac{m_i}{12}\left(3b_i^2 + l_i^2\right)$$

$$I_{i_{3,3}} = I_{i_{2,2}}$$

The off-diagonal terms are set to zero as there are no cross-products of inertia for the simple cylinder representation used to model the links. To specify a different inertia matrix, the user may input values directly.

Point masses may be added to each link if desired to create an arbitrary mass distribution. The addition of point masses requires that the total mass, center of gravity, and inertia matrix of the link be recalculated. The total mass is simply the sum of the link mass and all associated point masses

$$m_{i_{total}} = m_i + \sum_{n=1}^{\#\text{ of pts}} m_n$$

The new center of gravity is determined using

$$\underline{r}_{cg_{total}} = \frac{\sum_i m_i\, \underline{r}_{cg_i}}{\sum_i m_i}$$

Calculating a new inertia matrix is done using

$$[I_i]_{total} = [I_i] + m_i[I_i] + m_{pt}[I_{pt}]$$

$$[I_i] = (\underline{b}_i \cdot \underline{b}_i)[E] - \underline{b}_i \underline{b}_i^T$$

$$[I_{pt}] = (\underline{b}_{pt} \cdot \underline{b}_{pt})[E] - \underline{b}_{pt} \underline{b}_{pt}^T$$

where $[E]$ is the identity matrix and $\underline{b}_j$ is the vector from the new composite cg to the cg of j (link or point mass).

It should be noted that in addition to the three joint types mentioned earlier, "special joints" where the motion between adjacent joints may be constrained are also provided for. However, the use of any special joint would require a program modification that is not readily available at this time.

End-effectors. - The end-effector of a manipulator arm is modeled exactly as the link of a joint/link pair and includes the same provisions as links do for specifying arbitrary mass distributions.

Graphics. - ROBSIM has the capability to define detailed shapes for any component of the manipulator arm. Each part of the arm may be described as one of, or a combination of, any of the following shapes:

1) Cylinder;
2) Cone;
3) Rectangular solid;
4) Symmetric trapezoid;
5) Nonsymmetric trapezoid;
6) Triangular structure;
7) Data tablet structure;
8) Fillet;
9) Nonplanar entity.

These detailed geometries are used only for the Evans and Sutherland graphics display. All mass properties and arm motions are based on simple cylinder representations of the arm components. Figure 4 shows a detailed graphic representation of a manipulator arm.

Environment Creation/Modification

Definition of an environment simulates the workspace or surroundings of a manipulator system. Any components considered to be part of the environment are completely stationary once placed in the system. Consequently, no mass properties are ever defined and the environment is used only for the Evans and Sutherland graphic display. There is no effect on manipulator arm motion or any of the analysis capabilities. Environment definition is carried out by specifying the placement of detailed geometric components in the world coordinate system. The geometries available are:

1) Cylinder;
2) Cone;
3) Rectangular solid;
4) Symmetric trapezoid;
5) Nonsymmetric trapezoid;
6) Triangular structure;
7) Data tablet structure;
8) Fillet;
9) Obstacle entity.

To specify the placement of these components, each has its own local coordinate system whose x-axis is the centerline of the component. The origin of the local coordinate system is contained in the vector $\underline{r}_i$. The orientation of each component's local coordinate system is defined by a transformation matrix whose columns contain the direction cosines of the local x, y, and z axis to the world coordinate system axes.

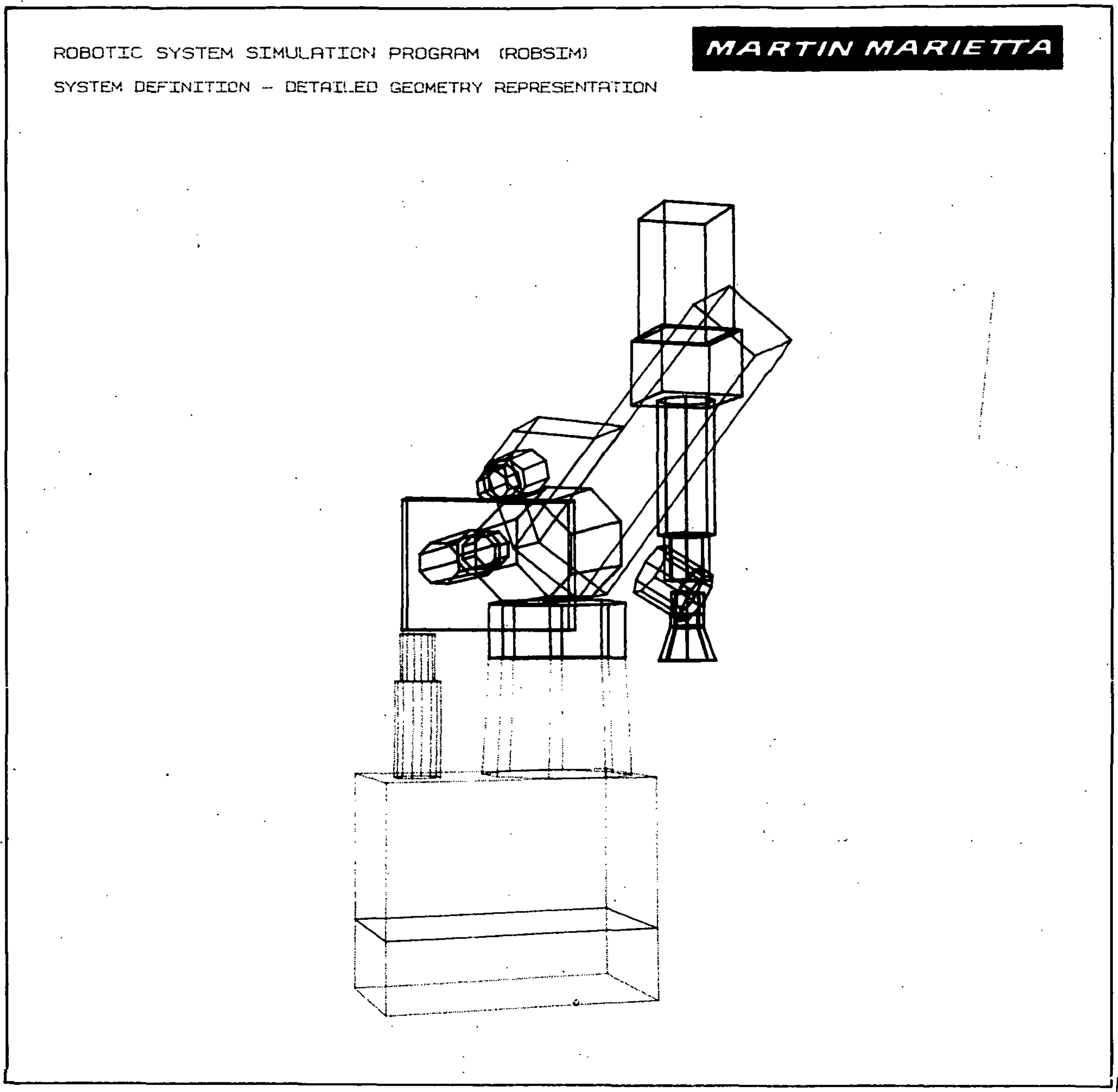

ROBOTIC SYSTEM SIMULATION PROGRAM (ROBSIM)
SYSTEM DEFINITION - DETAILED GEOMETRY REPRESENTATION
MARTIN MARIETTA

Figure 4. - Detailed graphics representation of manipulator arm.

Load Objects Creation/Modification

Load objects are similar to the environment in that they are used to simulate the workspace of a manipulator. Unlike components of the environment, load objects are not stationary but may be moved by a manipulator arm. Similar to arm link components, each load object is defined initially as a simple cylinder with a local coordinate system X_L, and with the centerline of the object along the local x-axis. The location of the origin of the X_i coordinate system is defined by the vector $\underline{r_L}$. The orientation of the local coordinate system is again defined by a transformation matrix whose columns are the direction cosines of the local x, y and z axes to the world coordinate system axes.

The center of mass of each load object is determined using the method described for manipulator links. A different center of mass may be specified by user definition of its x, y, and z coordinates in the X_L coordinate system. Inertia matrix calculations are also the same as for the links of a manipulator arm. Point masses may be added to a load object to create an arbitrary mass distribution. The location of the point mass is defined by the vector $\underline{_L r_{rt}}$. The new mass properties (total mass, center of gravity, and inertia matrix) are calculated by the same algorithms as used for the manipulator links.

The preceding paragraphs have described the mass properties for load objects. Evans and Sutherland graphic displays are available to portray the load objects. This capability allows the description of detailed geometries for each object. The detailed geometry definition is done by describing each load object as a group of one or more of the following geometry components:

1) Cylinder;
2) Cone;
3) Rectangular solid;
4) Symmetric trapezoid;
5) Nonsymmetric trapezoid;
6) Triangular structure;
7) Data tablet structure;
8) Fillet;
9) Nonplanar entity.

Each of these components has its own local coordinate system whose location and orientation is specified with respect to the load local coordinate system. Figure 5 shows the relation of component coordinate systems to the load local coordinate system.

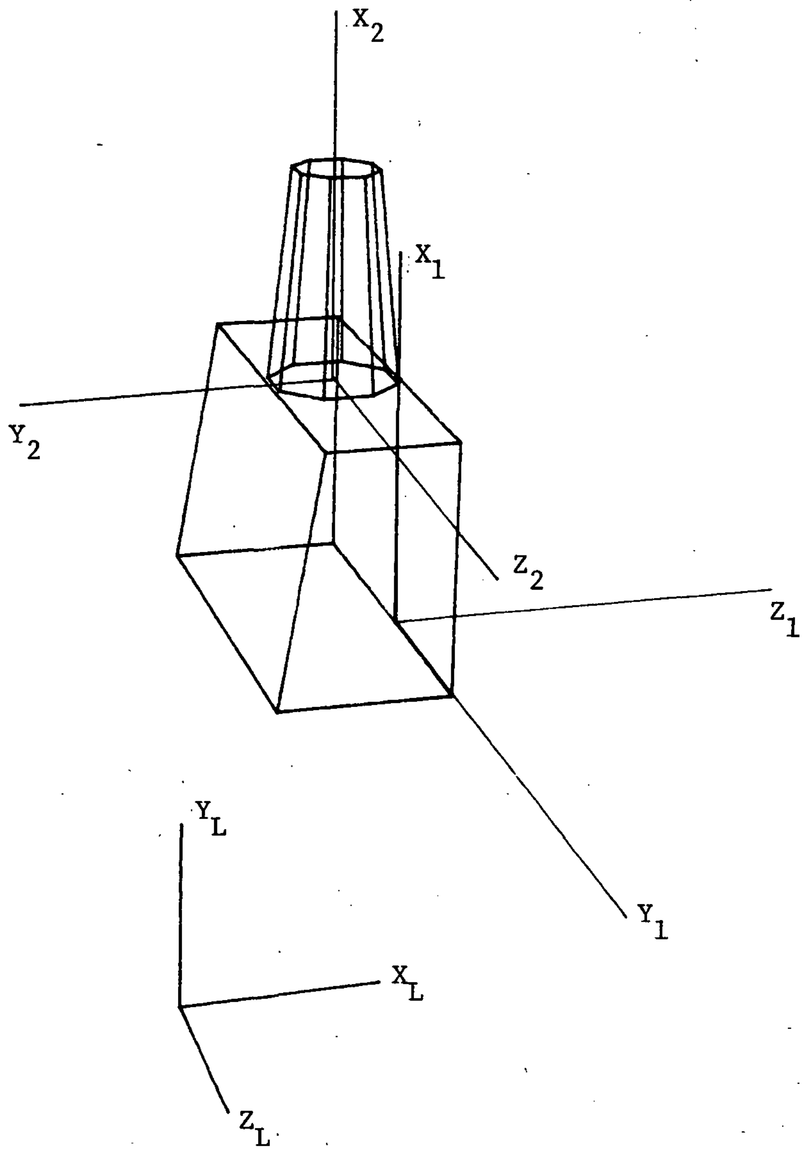


Figure 5. - Load local and component coordinate systems.

System Creation

Creation of a manipulator system brings together all of the items discussed up to this point—manipulator arms, an environment, and load objects. The system is put together with respect to a reference or world coordinate system. This is the same world coordinate system used previously for item definition. Adding an environment to the system places it at the same location and orientation as specified during environment creation.

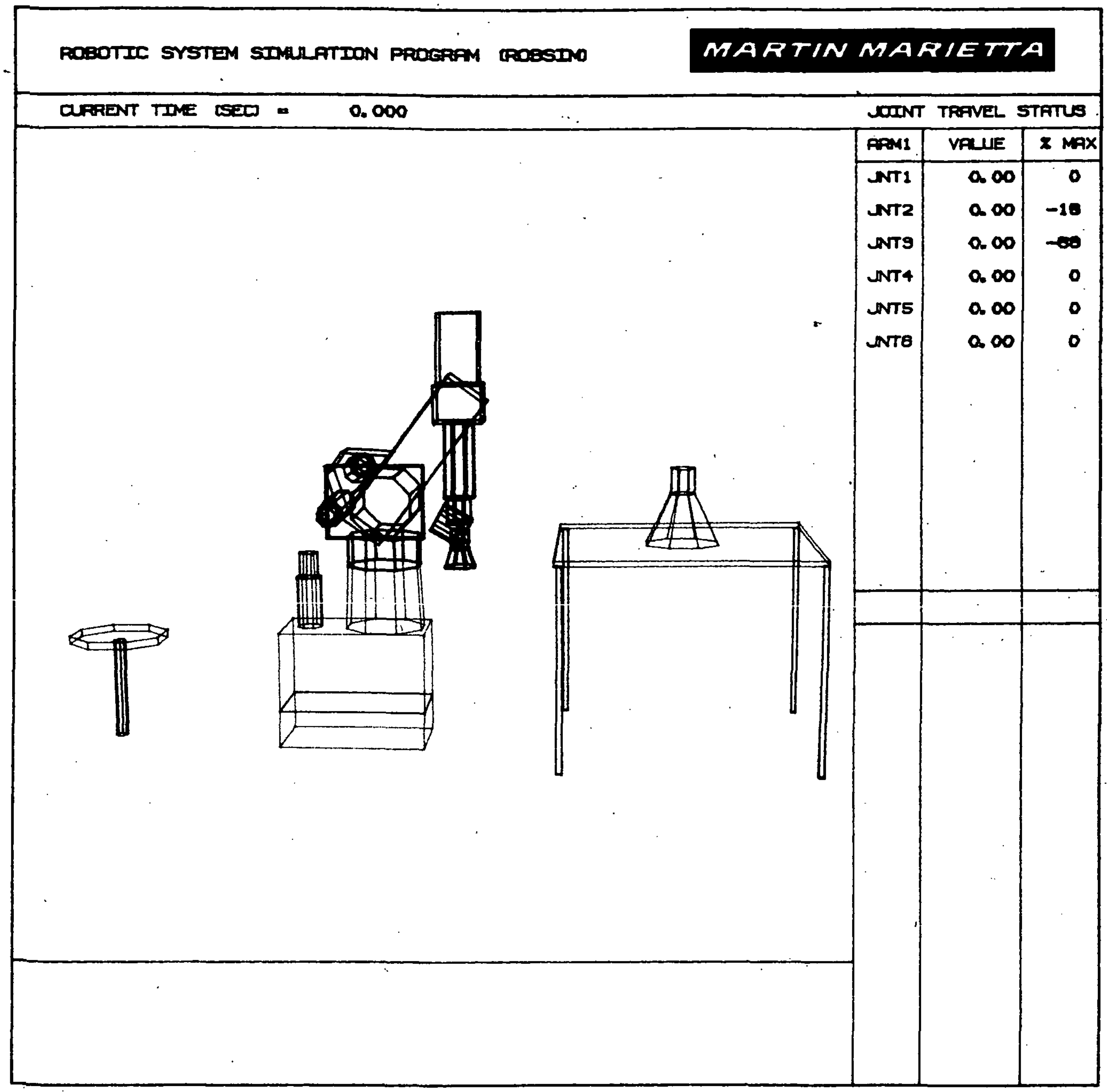
Placement of a manipulator arm in the system may be modified from the location and orientation called out during arm creation. The new location of the arm base is stored in the same vector $\underline{R}_b$. Location vectors for each joint are also updated. The new base orientation is a concatenation of the matrix $[P_b]$ specified during arm creation and the matrix $[{}_bP_{b'}]$, which specifies the new orientation with respect to the old orientation. The transformation matrix for the new base orientation with respect to the world is found using

$$[P_{b'}] = [P_b][{}_bP_{b'}]$$

The orientation of each joint in the rest of the manipulator arm is revised using the same procedure

$$[P_{i'}] = [P_i][{}_bP_{b'}]$$

Placement of load objects in the system is identical to manipulator arm placement. Each object may be placed in a new location and/or orientation. Figure 6 shows an Evans and Sutherland display of a system that includes one arm, a table as the environment, and two load objects



ROBOTIC SYSTEM SIMULATION PROGRAM (ROBSIM)

MARTIN MARIETTA

CURRENT TIME (SEC) = 0.000

JOINT TRAVEL STATUS

ARM1	VALUE	% MAX
JNT1	0.00	0
JNT2	0.00	-18
JNT3	0.00	-68
JNT4	0.00	0
JNT5	0.00	0
JNT6	0.00	0

Figure 6. - Complete system definition.

ANALYSIS CAPABILITY

Introduction

This section describes the analysis tools implemented in the ROBSIM package for investigating load and motion properties of general manipulator systems. Three types of analysis can be performed on the user-defined system:

1) Kinematic analysis;

2) Requirements analysis;

3) Response simulation.

Kinematic analysis involves determining positions and position derivatives (motion) of the manipulator links. It is important in its own right for investigating arm reachability, workpiece placement, task sequencing, obstacle avoidance, tool rate limits, etc; it also forms the basic step in all appropriate dynamic analysis formulations.

Requirements analysis and response simulation are the two dynamic analysis options available in the ROBSIM package. Requirements analysis, also referred to as "inverse dynamics" or "kinematically driven analysis," involves determining the operating forces and torques for a prescribed motion state of the manipulator. This provides actuator torque and sizing criteria, component stress levels, load-handling limits, feedforward compensation values for control, etc.

For response simulation, or "force-driven analysis," the actuator driving torques are specified (possibly in the form of a feedback control law) and the resulting motion trajectory of the manipulator is calculated. This option is especially useful for evaluating control strategies, task performance, interactions with the environment, etc.

The final part of this section describes the algorithms employed in the ROBSIM package for modeling end-effectors, in particular the parallel jaw gripper developed and in use at the Intelligent Systems Research Laboratory (ISRL) at NASA-LRC.

Some results obtained using these ROBSIM analysis tools are demonstrated in subsequent sections of this report. Table II lists the notation used in this section.

TABLE II.- NOTATION IN ANALYSIS CAPABILITIES DISCUSSION

$\underline{a}_i$	Acceleration of Special Point i
$\underline{a}_i^V$	Velocity-Related Acceleration of Special Point i, i.e., Acceleration When Joint Accelerations Are Zero
$[A]$	Effective Inertia Matrix in Equations of Motion
$\underline{b}$	Vector of Position- and Velocity-Related Torques in Equations of Motion
c_i	Contact Points between Peg and Gripper, i=1, 2
$\underline{e}_r$	Direction of Tangential Velocity of Constrained Point
f_{ci}	Force at Contact Point c_i
f_{gr}	Gripping Force
$\underline{f}_i$	Force at Point i
$\underline{f}_r$	Reaction Force at Constraint, with Magnitude f_r or $f_r^+ - f_r^-$
$\underline{g}$	Acceleration Due to Gravity
$\underline{h}_{ii}$	Vector from Origin of Link i to the Centroid of That Link*
$\underline{h}_{i,i+1}$	Vector from Origin of Link i to Origin of Link i+1*
i	Index, Generally 1, 2, ..., N+1 Indicating Link i, But Also b-Base, L-Load, gr-Gripper, p-End-Effector Reference Point, cg_i-Centroid of Link i
I_{em}	Effective Inertia of Motor and Drive
I_{gr}	Effective Inertia Associated with Gripping Motion
$[I_i]$	*Inertia Matrix of Second Moments of Inertia about Centroid for Link i*

$[J]$	Jacobian Matrix Relating Hand Motion to Joint Motion
$\underline{J}_j^i$	Vector of Jacobian Relating Motion of Link i to Joint j
K_{amp}	Amplifier Gain in Gripper Drive
K_{ec}	Effective Coulomb Friction Coefficient
L_{emf}	Effective Back-emf Constant for Motor
K_{es}	Effective Static Friction Coefficient
K_{et}	Effective Torque Constant for Motor
K_{ev}	Effective Viscous Friction Coefficient
$[K_{sp}]$	Matrix of Spring Constants Relating End-Effector Force to Position
K_{Tach}	Coefficient Relating Tachometer Voltage to Gripper Velocity
ℓ_p	Length of Peg
M_i	Mass of Link i
n_a	Actuator Gear Ratio
$\underline{n}_r$	Vector Normal to Constraint Surface
N	Number of Joints
N+1	Refers to End-Effector
$[P_i]$	Matrix of Direction Cosines Specifying Orientation of ith Coordinate System with Respect to World Coordinate System
$[{}_iP^{ref}_{i+1}]$	*Transformation Matrix from Coordinate System i to the Reference* Position of Coordinate System i+1

TABLE II. - (CONCL)

$\underline{r}$	Vector Locating a Point of Interest in World Coordinate System*
$\underline{r}_{cg_i}$	Location of Center of Gravity of Link i in World Coordinate System
$\underline{r}_p$	Location of End-Effector Reference Point or End of Peg
R_a	Motor Armature Resistance
$\underline{R}_i$	Vector Locating Origin of ith Reference
$\dot{q}$	Generalized Velocity Coordinate Used in Rate Control
$\underline{Q}$	Generalized Momentum in Joint Coordinates
s	Laplace Variable
$\underline{S}_i$	Unit Vector along Axis of Link i, Referenced to World Coordinate System
t	Time
$\underline{t}_i$	Reaction Torque at Joint i
$\underline{T}_I$	Generalized Impulse in Joint Coordinates
$u(t)$	Scalar Representing Proportion of Motion Segment Traversed by Time t
v_{dac}	Control Voltage Driving Gripper Operation
$\underline{V}_i$	Velocity of Reference Point i
v_i	Voltage Driving the Motor for Joint i
$\underline{w}_f$	Vector Transforming Constraint Friction Force to Joint Coordinates
$\underline{w}_r$	Vector Transforming Constraint Reaction Force to Joint Coordinates
$\underline{X}_i$	Coordinate System Associated with Component i
$\underline{\alpha}_i$	Angular Acceleration of Body i
$\underline{\alpha}_i^v$	Velocity-Related Angular Acceleration of Body i
$[\Delta P]$	Change in an Orientation Matrix
$\Delta\underline{r}_p$	Change in Position of End-Effector Reference Point
$\Delta\underline{R}_{N+1}$	Error in Position of Tool Origin Measured from Current to Desired Position
$\Delta\underline{\phi}$	Rotation Vector Representing a Change in Orientation
$\Delta\underline{\theta}$	Small Displacement of Joints
ϕ	Rotation Angle Corresponding to Change in Orientation
$\underline{\theta}$	Relative Joint Displacements - Used As Generalized Coordinates
θ_i	Relative Joint Displacement at Joint i
θ_{gr}	Gripper Displacement Angle
μ	Coefficient of Friction at Constraint Contact Surface
ρ	Special Implicit Constraint Function
σ	Signum Function, Equal to ± 1
$\underline{\tau}$	Vector of Generalized Torques
$\underline{\omega}_i$	Angular Velocity of Body i

*A subscript i preceding these vectors indicates that the vector is specified relative to the local coordinate system i.

The kinematic and dynamic analysis tools implemented in ROBSIM are based on a rigid-link model of serial, open-loop kinematic chains with one-degree-of-freedom joints. This subsection first describes the forward kinematics solution. This is, given the vector $\underline{\theta}$ of relative joint displacements and their first and second time derivatives $(\dot{\underline{\theta}}, \ddot{\underline{\theta}})$ for a specified arm geometry, find the positions, orientations, velocities and accelerations of all links and points of interest in the device. Then methods for generating the time trajectories of the joint displacements $(\underline{\theta}(t), \dot{\underline{\theta}}(t), \ddot{\underline{\theta}}(t))$, particularly for providing task-oriented operation of the manipulator, are discussed.

<u>Link positioning</u>. - Determining the position and orientation of each link is a fundamental step in the analysis of complex mechanisms such as manipulators. Figure 7 presents a simple graphical representation of a serial manipulator arm. As described in the preceding section of this report, a local Cartesian coordinate system $\underline{X}_i$ is associated with each link of the arm. The complete position of the link is specified by the coordinate position of one point in the link and the orientation of the link.

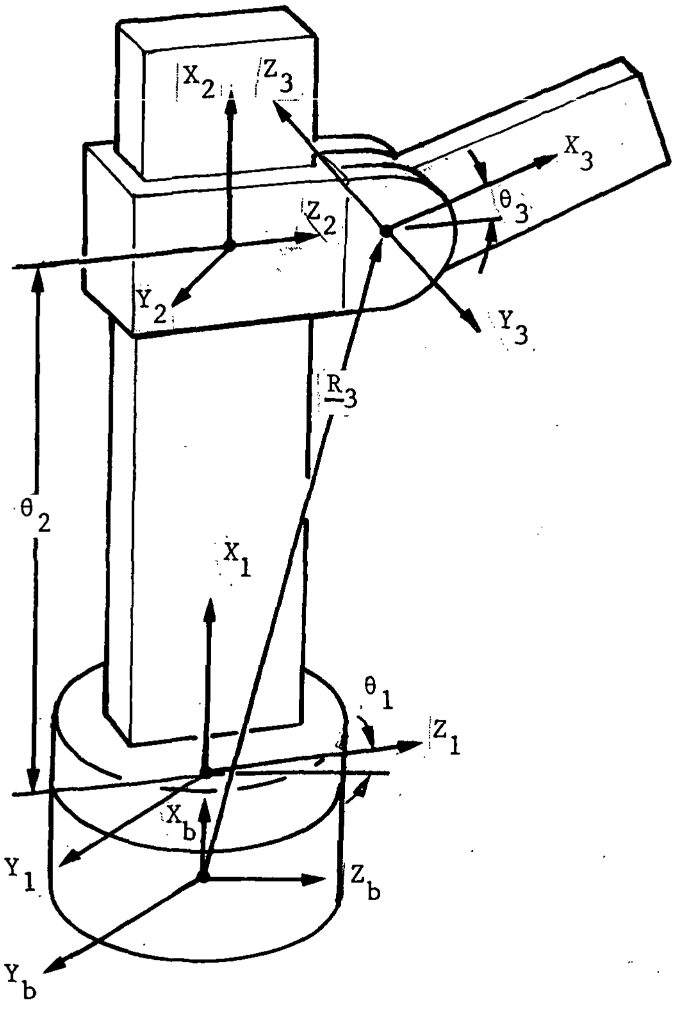


Figure 7.-
Kinematic representation of a serial manipulator.

The vector $\underline{R}_i$ contains the components of the location, in an inertial world coordinate system, of the origin of the $\underline{X}_i$ reference (Fig. 7). The orientation of the link is defined by the 3x3 rotation matrix of direction cosines $[P_i]$. The matrix $[P_i]$ is an orthogonal matrix; each column contains a unit vector specifying the direction, in world coordinates, of the corresponding local reference axis. Although this method for specifying orientation contains an overabundence of parameters (nine parameters, of which only three are independent), it is convenient for several reasons, including:

1) The $[P_i]$ matrix forms the basis for transforming vectors from one coordinate system to another;

2) It contains a column specifying the direction of the joint axis, which is needed for subsequent computations;

3) It is readily determined uniquely from other orientation specifications.

As an example of the coordinate transform, suppose $_i\underline{r}$ specifies the direction of a vector in terms of the local coordinate system $\underline{X}_i$. The direction of this vector in the world coordinate system is given by

$$\underline{r} = [P_i]\,{}_i\underline{r}$$

Recall that the preceding subscript indicates the coordinate reference for the vector, with the default value W for the world reference. The full point transformation is obtained by adding the vector to the origin of $\underline{X}_i$. That is, if $_i\underline{r}$ specifies the local coordinates of a point fixed in link i, then the coordinates of this point in the world reference are*

$$\underline{r} = \underline{R}_i + [P_i]\,{}_i\underline{r}$$

Note that since $[P_i]$ is orthogonal,

$$[P_i]^{-1} = [P_i]^T$$

where the T superscript indicates transpose; also, successive transformations are produced by matrix multiplication

$$[{}_kP_i] = [{}_kP_j]\;[{}_jP_i]$$

so transformations between arbitrary sets of coordinates are readily obtained.

A recursive method is employed for finding the link positions; it is computationally efficient and readily programmed. The orientation matrices for the links are successively computed starting with the base and proceeding along the serial chain to the free end of the manipulator. Suppose that the rotation matrix $[P_i]$ for link i is known. The corresponding values for link i+1 are then given by

$$[P_{i+1}] = [P_i]\;[{}_iP_{i+1}]$$

*This transformation is often represented in homogeneous coordinates using a 4x4 displacement matrix [A] (See [Paul 1981a])

$$[A] = \left[\begin{array}{ccc|c} & [P_i] & & \underline{R}_i \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

The orientation matrix $[{}_iP_{i+1}]$ between successive links i and i+1 consists of two components: (1) a fixed link rotation matrix $[{}_iP_{i+1}^{ref}]$ that defines the transformation from coordinate system i to the reference position of system i+1 and is established during system definition, and (2) a joint rotation matrix $[{}_iP_{i+1}]$ that varies with the joint displacement θ_{i+1} for rotating joints. Because hinge joints rotate about the local Y-axis and swivel joints rotate about the local X-axis, the joint rotation matrices are defined by the following

$$[{}_iP_{i+1}^{\theta}] = \begin{bmatrix} \cos\theta_{i+1} & 0 & \sin\theta_{i+1} \\ 0 & 1 & 0 \\ -\sin\theta_{i+1} & 0 & \cos\theta_{i+1} \end{bmatrix} \quad \text{(Hinge Joint i+1)}$$

$$[{}_iP_{i+1}^{\theta}] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_{i+1} & -\sin\theta_{i+1} \\ 0 & \sin\theta_{i+1} & \cos\theta_{i+1} \end{bmatrix} \quad \text{(Swivel Joint i+1)}$$

$$[{}_iP_{i+1}^{\theta}] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{(Sliding Joint i+1)}$$

The only place in the formulation that trancendental functions are evaluated is in setting up these joint rotation matrices. The full recursive relation for reference i+1 is computed as

$$[P_{i+1}] = [P_i]\,[{}_iP_{i+1}^{ref}]\,[{}_iP_{i+1}^{\theta}]$$

Successively applying this relation starting from a known position of the base, each link's orientation matrix is obtained.

Once these rotation matrices are determined, vectors are transformed from local to world coordinates and the locations of the coordinate origins are recursively computed. For rotating joints i+1, the relation is

$$\underline{R}_{i+1} = \underline{R}_i + \underline{h}_{i,i+1} \quad \text{(Rotating Joint i+1)}$$

where

$$\underline{h}_{i,i+1} = [P_i]\,{}_i\underline{h}_{i,i+1}$$

and ${}_i\underline{h}_{i,i+1}$ is the vector, in local coordinates, from link i to link i+1, which is specified during system definition. If joint i+1 is a sliding joint, an additional term for the displacement along the joint axis (local x-axis) must be added (see Fig. 7)

$$\underline{R}_{i+1} = \underline{R}_i + \underline{h}_{i,i+1} + [P_{i+1}] \begin{Bmatrix} \theta_{i+1} \\ 0 \\ 0 \end{Bmatrix} \quad \text{(Sliding Joint i+1)}$$

Again, successive application of these recursive relations yields the location of each local coordinate reference.

Note that once the position of each link's reference $\underline{X}_i$ is available, the world coordinates for any point in the link can be found using the point transformation given earlier. For example, ${}_i\underline{r}_{cgi}$ represents the local coordinates of the centroid of link i and is established during system definition. The instantaneous location of this centroid in terms of the world reference is given by

$$\underline{r}_{cg_i} = \underline{R}_i + [P_i]\, {}_i\underline{r}_{cg_i}$$

Link and point velocities. - The velocity of a rigid body is conveniently specified as the translational velocity of some point in the body along with the angular velocity of the body. A recursive method based on classical rigid-body kinematics is employed to calculate the velocities and accelerations of the individual manipulator links. The method has received much attention in the recent literature on manipulator dynamics (e.g., see [Orin 1979], [Luh 1980], [Walker 1982]), and was chosen for its efficiency, simplicity, and ease of programming.

Let $\underline{\omega}_i$ represent the angular velocity of link i referenced to the world coordinate system. Assuming $\underline{\omega}_i$ is known, the angular velocity $\underline{\omega}_{i+1}$ of the next link is readily given as the sum of $\underline{\omega}_i$ plus the angular velocity of link i+1 relative to link i, $\underline{\omega}_{i+1/i}$. If joint i+1 is a rotating joint, then this relative angular velocity is a vector of magnitude $\dot{\theta}_{i+1}$ about the axis of rotation, i.e.,

$$\underline{\omega}_{i+1/i} = \dot{\theta}_{i+1}\, \underline{S}_{i+1} \quad \text{(Rotating Joint i+1)}$$

Here, $\underline{S}_{i+1}$ is a unit vector along the joint axis and is given in terms of the link's orientation as

$$\underline{S}_{i+1} = [P_i] \begin{Bmatrix} 0 \\ 1 \\ 0 \end{Bmatrix} \quad \text{(Hinge Joint i+1)}$$

$$\underline{S}_{i+1} = [P_i] \begin{Bmatrix} 1 \\ 0 \\ 0 \end{Bmatrix} \quad \text{(Swivel or Sliding Joint i+1)}$$

Therefore, starting with the specified angular velocity for the base, the angular velocity of each link is obtained by successive application of the recursion relations

$$\underline{\omega}_{i+1} = \underline{\omega}_i + \dot{\theta}_{i+1} \underline{S}_{i+1} \quad \text{(Rotating Joint i+1)}$$

$$\underline{\omega}_{i+1} = \underline{\omega}_i \quad \text{(Sliding Joint i+1)}$$

Subsequently, the translational velocity $\underline{V}_{i+1}$ of the origin of the $\underline{X}_{i+1}$ reference can be represented in terms of the translational and rotational velocities of link i. If joint i+1 is a rotational joint, the relation is

$$\underline{V}_{i+1} = \underline{V}_i + \underline{\omega}_i \times (\underline{R}_{i+1} - \underline{R}_i) \quad \text{(Rotating Joint i+1)}$$

while an additional term must be included for the sliding joint case

$$\underline{V}_{i+1} = \underline{V}_i + \underline{\omega}_i \times (\underline{R}_{i+1} - \underline{R}_i) + \dot{\theta}_{i+1} \underline{S}_{i+1} \quad \text{(Sliding Joint i+1)}$$

Recall that the vector from joint i to joint i+1, which is used in the vector crossproduct term in these equations, is given by

$$\underline{R}_{i+1} - \underline{R}_i = \underline{h}_{i,i+1} \quad \text{(Rotating Joint i+1)}$$

$$\underline{R}_{i+1} - \underline{R}_i = \underline{h}_{i,i+1} + \theta_{i+1} \underline{S}_{i+1} \quad \text{(Sliding Joint i+1)}$$

In addition, if the velocity $\underline{V}_p$ of any point P fixed in link i is desired, it is then obtained from

$$\underline{V}_p = \underline{V}_i + \underline{\omega}_i \times (\underline{r}_p - \underline{R}_i)$$

For a specified velocity $\underline{V}_b$ of the manipulator base, the application of these formulas provides values for the velocities of each link and all points of interest directly from the position results.

<u>Link and point accelerations</u>. - In a similar manner, the accelerations of the manipulator links can be solved for by recursive application of the second-order motion derivatives for rigid bodies. Let $\underline{\alpha}_i$ represent the angular acceleration of link i and $\underline{a}_i$ the translational acceleration of the corresponding reference origin. The acceleration equations are as follows:

$$\underline{\alpha}_{i+1} = \underline{\alpha}_i + \underline{\omega}_i \times \dot{\theta}_{i+1} \underline{S}_{i+1} + \ddot{\theta}_{i+1} \underline{S}_{i+1} \quad \text{(Rotating Joint i+1)}$$

$$\underline{\alpha}_{i+1} = \underline{\alpha}_i \quad \text{(Sliding Joint i+1)}$$

$$\underline{a}_{i+1} = \underline{a}_i + \underline{\omega}_i \times [\underline{\omega}_i \times (\underline{R}_{i+1} - \underline{R}_i)] + \underline{\alpha}_i \times (\underline{R}_{i+1} - \underline{R}_i) \quad \text{(Rotating Joint i+1)}$$

$$\underline{a}_{i+1} = \underline{a}_i + \underline{\omega}_i \times [\underline{\omega}_i \times (\underline{R}_{i+1} - \underline{R}_i)] + \underline{\alpha}_i \times (\underline{R}_{i+1} - \underline{R}_i)$$
$$+ 2\underline{\omega}_i \times \dot{\theta}_{i+1} \underline{S}_{i+1} + \ddot{\theta}_{i+1} \underline{S}_{i+1} \quad \text{(Sliding Joint i+1)}$$

Again, the accelerations $\underline{a}_b$ and $\underline{\alpha}_b$ of the base are presumed known, and the acceleration $\underline{a}_p$ of any point of interest in link i can be readily obtained:

$$\underline{a}_p = \underline{a}_i + \underline{\omega}_i \times [\underline{\omega}_i \times (\underline{r}_p - \underline{R}_i)] + \underline{\alpha}_i \times (\underline{r}_p - \underline{R}_i)$$

In particular, the acceleration $\underline{a}_{cgi}$ of each link's centroid is calculated for use in the subsequent dynamic analysis.

Note that all vectors used in the preceding equations are expressed in terms of the world reference system. Any other coordinate selection is possible; the only constraint is that a consistent set of coordinates must be used throughout each equation.

End-effector motion. - While the kinematics and dynamics of serial manipulators are most conveniently evaluated in terms of the joint displacements and rates, it is often more useful to prescribe the motion of the terminal link ("end-effector," "tool," or "hand") of the device, especially for task-oriented operation. The algorithms employed in the ROBSIM package for transforming end-effector motion specifications into joint motion states are described here.

The inverse positioning problem, that of finding a set of joint displacements $\underline{\theta}$ corresponding to a prescribed end-effector position can be exceedingly complex* and analytic solution methods are not readily generalizable. However, commercial robots generally have special geometries, resulting in much simpler positioning solutions (e.g., see [Duffy 1979], [Paul 1981b]). Special routines can be programmed for certain geometries or classes of geometries. For generality, an iterative positioning routine is implemented in ROBSIM; because it involves an extension of the velocity results, it is described after the velocity relations are presented.

*For example, the solution for a six-degree-of-freedom arm can involve finding roots to a polynomial of up to 32nd order (see [Duffy 1981], [Duffy 1980]). Methods for analytically handling the positioning of redundant manipulators are not even available.

A method for recursively computing the end-effector velocity from the joint rates was described previously. For a given position $\underline{\theta}$ of the manipulator, the velocity ($\underline{V}_{N+1}$, $\underline{\omega}_{N+1}$) of the end-effector link N+1 can also be written explicity in terms of the joint velocities as

$$\begin{Bmatrix} \underline{V}_{N+1} \\ \underline{\omega}_{N+1} \end{Bmatrix} = [J(\underline{\theta})]\ \underline{\dot{\theta}} \qquad (*)$$

Here, $[J(\theta)]$ is the Jacobian matrix relating the end-effector motion to joint motion. Column i of the 6xN matrix [J] is readily expressed in terms of the position results as follows (see [Whitney 1972], [Thomas 1982]):

$$\underline{J}_i \equiv \begin{Bmatrix} \dfrac{\partial \underline{V}_{N+1}}{\partial \dot{\theta}_i} \\ \hline \dfrac{\partial \underline{\omega}_{N+1}}{\partial \dot{\theta}_i} \end{Bmatrix} = \begin{Bmatrix} \underline{S}_i \times (\underline{R}_{N+1} - \underline{R}_i) \\ \hline \underline{S}_i \end{Bmatrix} \qquad \text{(Rotating Joint i)}$$

$$\underline{J}_i = \begin{Bmatrix} \underline{S}_i \\ 0 \\ 0 \\ 0 \end{Bmatrix} \qquad \text{(Sliding Joint i)}$$

(The motion of some reference point other than the tool reference origin can be used by replacing $\underline{V}_{N+1}$ and $\underline{R}_{N+1}$ by $\underline{V}_p$ and $\underline{r}_p$ in these equations.)

For a specified end-effector velocity, Eq. (*) above represents a simultaneous set of six scalar equations linear in the N unknown joint velocities θ_i. These equations will result in either a unique solution, an infinite number of solutions, or no exact solutions; they can be solved by standard techniques of linear algebra. For example, using the pseudo-inverse $[J]^{-1*}$ of the Jacobian

$$\underline{\dot{\theta}} = [J]^{-1*} \begin{Bmatrix} \underline{V}_{N+1} \\ \underline{\omega}_{N+1} \end{Bmatrix}$$

produces a solution that minimizes the quadratic norm of the error between the desired and actual end-effector velocities (see [Lowrie 1982]). In addition, where multiple solutions exist the quadratic norm of the joint velocities is minimized. This method works quite well in general, although it can produce undesirable results when the arm is near a kinematic singularity.

Kinematic singularities are special configurations of the manipulator for which the rank of the Jacobian matrix decreases by one or more (e.g., see [Sugimoto 1982]). In these configurations, the end-effector loses degrees of freedom of instantaneous motion. When the manipulator approaches such configurations, the solution techniques previously described can result in excessively large velocity specifications for some of the joints. To prevent this, the individual joint velocities are constrained by the limits specified during system definition

$$|\dot{\theta}_i| \leq \dot{\theta}_{i,\max}$$

The joint velocities are scaled to satisfy this criterion. In the case of redundant manipulators (including instantaneous redundancy while in a singular configuration), linear programming is employed to minimize the magnitude of the largest joint velocity, i.e.,

$$\text{minimize } \{\underset{i}{\text{maximum}} \, (|\dot{\theta}_i|/\dot{\theta}_{i,\max})\}$$

This method produces realistic values for the joint velocity specifications and prevents the erratic behavior in the iterative positioning algorithm that can result without these modifications.

Once the joint velocities have been obtained from the end-effector motion specifications, the joint accelerations can be evaluated. Finite differencing may be employed:

$$\ddot{\theta}_i = \frac{\dot{\theta}_i(t) - \dot{\theta}_i(t - \Delta t)}{\Delta t}$$

or the joint accelerations can be determined from the end-effector motion specifications using the relations

$$\begin{Bmatrix} \underline{a}_{N+1} \\ \underline{\alpha}_{N+1} \end{Bmatrix} = [J]\underline{\ddot{\theta}} + \begin{Bmatrix} \underline{a}^{V}_{N+1} \\ \underline{\alpha}^{V}_{N+1} \end{Bmatrix}$$

Here, $\underline{a}^{V}_{N+1}$ and $\underline{\alpha}^{V}_{N+1}$ are the velocity-related accelerations of the end-effector. These can be obtained from the recursive kinematic method discussed previously by setting $\ddot{\theta}_i = 0$ in those earlier equations. As with the velocity transformation, the matrix of coefficients is the Jacobian, and the same linear equation solution procedures previously described can be applied to solve for $\underline{\ddot{\theta}}$.

Returning to the problem of finding the joint positions $\underline{\theta}$ for a prescribed end-effector position, an iterative method is implemented in ROBSIM. The algorithm uses the solution of a set of linear equations, where the Jacobian again forms the coefficient matrix, and is described in [Whitney 1972]. Based on the current position and desired position of the end-effector, a six-dimensional position error vector is obtained. This error vector is used to solve for a vector of joint corrections $\Delta\underline{\theta}$ from

$$\begin{Bmatrix} \Delta\underline{R}_{N+1} \\ \Delta\underline{\phi} \end{Bmatrix} = [J]\Delta\underline{\theta}$$

where $\Delta\underline{R}_{N+1}$ is the translational position error and $\Delta\underline{\phi}$ represents a rotation vector (the magnitude gives the angle of rotation and the direction specifies the axis) that rotates the end-effector from the current to the desired orientation. A method for solving for the vector $\Delta\underline{\phi}$ is described in the next subsection. Applying the special solution methods previously discussed allows limits to be placed on the size of the change $\Delta\underline{\theta}_i$ in the joint angles. This improves convergence of the positioning method for large initial errors or when operating near singularities.

After solving for $\Delta\underline{\theta}$, the joint displacements are updated,

$$\underline{\theta}_{new} = \underline{\theta}_{old} + \Delta\underline{\theta}$$

the current end-effector position is recalculated, and the process is repeated until the end-effector position converges to the desired value. Because this is an iterative method, only a single position solution will result (depending on the initial values of $\underline{\theta}$), although the manipulator geometry may have multiple configurations corresponding to the same end-effector position. To date, testing has shown the method to be quite robust as long as reasonable limits (e.g., 0.1 radian) are placed on the changes in joint displacement per iteration step.

Task-oriented motion specification. - This section has presented methods for determining link motions from joint motion states and for finding joint motions that produce end-effector trajectories. To make task programming more convenient, a user-interface is implemented that allows interactive specification of a sequence of operations and robot motions. The motions generated from these specifications are described in the following paragraphs.

The overall task specification is divided into motion segments. The following options are available for defining the motion within each individual time segment:

1) Rate control,
   a) Joint motion,
   b) End-effector motion (world coordinates and local end-effector coordinates);

2) Position control,
   a) Joint motion,
   b) End-effector motion (world coordinates).

For rate control motion, the velocities of the individual position coordinates are specified as polynominal functions of time

$$\dot{q} = a_n t^n + a_{n-1} t^{n-1} + \ldots + a_1 t + a_0$$

where $\dot{q}$ is the rate of some position component and the a_i are user-specified coefficients. For rate control of joint motion, $\dot{q}$ corresponds to the individual joint velocities, $\dot{\theta}_i$. Therefore, the joint velocities at any time during the segment are determined by evaluating the rate polynomials for the given time, t. Furthermore, the polynomials can be symbolically differentiated and the joint accelerations found by evaluating

$$\ddot{q} = na_n t^{n-1} + (n-1)a_{n-1} t^{n-2} + \ldots + a_1$$

The joint displacements are updated using Euler integration of the velocity results*

$$\theta_i\ (t + \Delta t) = \theta_i(t) + \Delta t * \dot{\theta}_i(t)$$

When rate control of end-effector motion is specified, the $\dot{q}$ polynominals correspond to the six components of end-effector velocity, $\underline{V}_p$ and $\underline{\omega}_{N+1}$.

(A different tool reference point P can be used for each motion segment.) At time t, the rate polynominals are evaluated to determine the end-effector velocity. These velocity vectors can be defined in either world coordinates or instantaneous end-effector coordinates. If defined in local end-effector coordinates, the vectors are transformed into world coordinates by premultiplying each by $[P_{N+1}]$. The joint velocities are then solved for using the Jacobian inversion method described previously. The joint accelerations are obtained from these velocities using the backwards finite-differencing equations, and the joint displacements are evaluated, as before, from the Euler integration equation.

Position control segments provide coordinated motion between user-specified trajectory endpoints. Each position component is scaled so they all reach their final values simultaneously. For example, for joint position control, the initial joint displacements $\underline{\theta}(t_0)$ are known from the manipulator's current location and the final displacements $\underline{\theta}(t_f)$ are specified. The joint positions at any intermediate time during the motion segment are then given by

$$\underline{\theta}(t) = \underline{\theta}(t_0) + u(t)\ \Delta\underline{\theta}$$

where

$$\Delta\underline{\theta} = \underline{\theta}(t_f) - \underline{\theta}(t_0)$$

*Although symbolic integration of the rate polynomial could be used for this case, the Euler integration method must be employed for the end-effector rate control specification.

and u(t) is a scaling value equal to the fraction of the total motion completed at time t,

$$u(t_0) = 0$$

$$u(t_f) = 1$$

A constant-acceleration constant-velocity constant-deceleration profile is currently implemented. Figure 8 illustrates the value of u(t) and its first and second-order derivatives during the motion segment. Different velocity profiles for the motion can be readily implemented by modifying the routines that evaluate u. The joint velocities and accelerations during the motion are given by

$$\underline{\dot{\theta}}(t) = \dot{u}(t)\ \Delta\underline{\theta}$$

$$\underline{\ddot{\theta}}(t) = \ddot{u}(t)\ \Delta\underline{\theta}$$

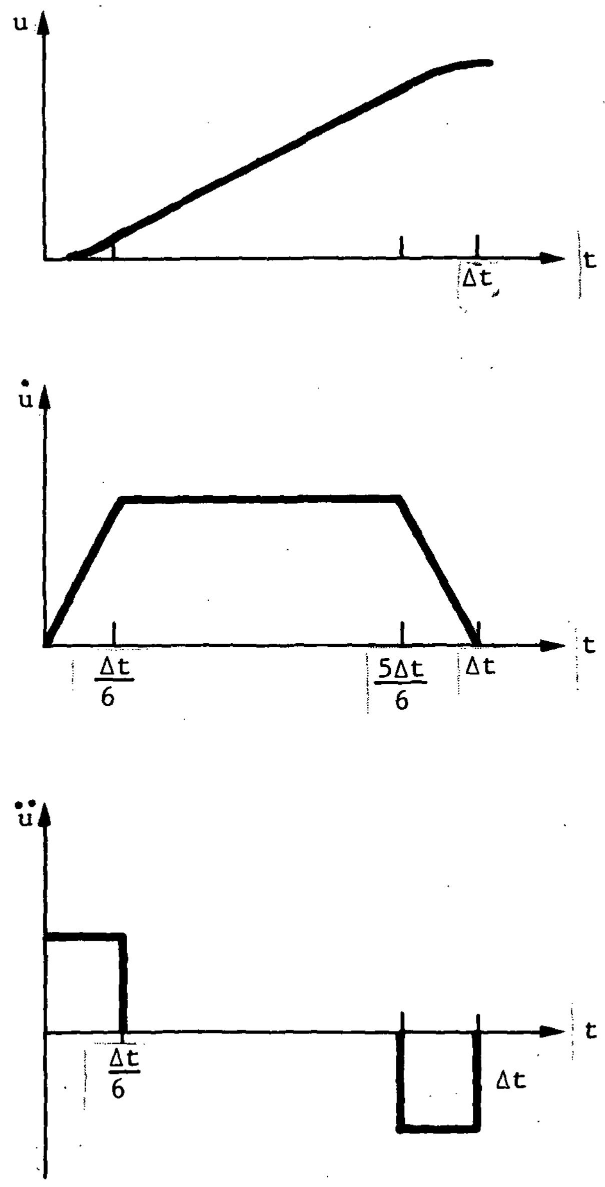


Figure 8.-
Constant-acceleration constant-velocity constant-deceleration profile.

Position control of the end-effector motion is the most popular type of motion specification. At the beginning of the motion segment, a six-dimensional vector representing the change in end-effector position and orientation is calculated. The first three components, $\Delta \underline{r}_p$, define the translation of the end-effector reference point during the motion, and the last three components, $\Delta \underline{\phi}$, define an end-effector rotation that produces the desired change in orientation. The method for calculating this rotation vector is described in the following discussion.

Recall that each column of the orientation matrix $[P_{N+1}]$ defines the direction of the corresponding local reference axis. The change in direction of these unit vectors are the columns of $[\Delta P]$ where

$$[\Delta P] = [P_{N+1}(t_f)] - [P_{N+1}(t_0)]$$

Because the rotation axis must be perpendicular to the changes in these unit vectors, its direction can be calculated as the crossproduct of any two columns* of $[\Delta P]$. The magnitude of the rotation is found using

$$\Delta\phi = \cos^{-1}\left(1 + \frac{\mathrm{tr}([P_{N+1}]^T\,[\Delta P])}{2}\right)$$

where tr () indicates the trace of the matrix. This equation has two solutions; the one corresponding to the direction chosen for the rotation axis can readily be evaluated.

As in the joint motion case, the position values at intermediate times during the motion are determined using the scaling factor u(t):

$$\underline{r}_p(t) = \underline{r}_p(t_0) + u(t)\ \Delta\underline{r}_p$$

$$\phi(t) = u(t)\ \Delta\phi$$

The orientation of the end effector during the motion segment is given in terms of ϕ as

$$[P_{N+1}(t)] = [P_{N+1}(t_0)]\ [DP(t)]$$

$$[DP(t)] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \sin\phi \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} + (1 - \cos\phi) \begin{bmatrix} 1-a_1^2 & a_2a_1 & a_3a_1 \\ a_1a_2 & 1-a_2^2 & a_3a_2 \\ a_1a_3 & a_2a_3 & 1-a_3^2 \end{bmatrix}$$

*The two columns of $[\Delta P]$ of largest magnitude are used so special cases are handled automatically.

where a_1, a_2 and a_3 are the components, in end-effector coordinates, of the unit vector along the rotation axis ϕ. With the end-effector position determined, the inverse positioning algorithm described earlier is used to find the joint displacements $\underline{\theta}$. Then the joint velocities $\underline{\dot{\theta}}(t)$ can be obtained from the end-effector velocities

$$\begin{Bmatrix} \underline{V}_p \\ \underline{\omega}_{N+1} \end{Bmatrix} = \dot{u}(t) \begin{Bmatrix} \Delta \underline{r}_p \\ \Delta \underline{\phi} \end{Bmatrix}$$

The joint accelerations are computed by the backwards difference method.

The user interface also provides for the specification of such nonmotion steps as GRASP, WAIT, etc. These are described in the ROBSIM User's Guide, Appendix A.

Requirements Analysis

Once the arm motion is established and the positions, velocities and accelerations of the individual links are determined, the reaction forces and torques throughout the manipulator structure can be readily evaluated using classical rigid-body dynamics. As in the kinematic analysis, ROBSIM employs a recursive method for dynamic requirements analysis because of the method's efficiency and tractability (see [Luh 1980], [Hollerbach 1981]). For this analysis, the forces and torques are successively computed starting from the terminal link of the arm and proceeding back to the manipulator base.

Joint reaction forces and torques. - The reaction force $\underline{f}_i$ and torque $\underline{t}_i$ at joint i are combinations of the reactions on link i at joint i+1, inertial loads caused by the acceleration of link i, and any other loads applied to the link. Assuming that the only external load applied to the intermediate links are the gravity forces acting through their centroids, the recursive equations for the joint reactions can be written

$$\underline{f}_i = \underline{f}_{i+1} + m_i \left(\underline{a}_{cg_i} + \underline{g}\right)$$

$$\underline{t}_i = \underline{t}_{i+1} + (\underline{R}_{i+1} - \underline{R}_i) \times \underline{f}_{i+1} + \underline{h}_{ii} \times m_i \left(\underline{a}_{cg_i} + \underline{g}\right) + [I_i]\, \underline{\alpha}_i + \underline{\omega}_i \times [I_i]\, \underline{\omega}_i$$

The new symbols in these equations are:

m_i - mass of link i;

$\underline{a}_{cg_i}$ - acceleration of centroid of link i;

$\underline{g}$ - acceleration due to gravity;

$[I_i]$ - inertia matrix of second moments for link i about its centroid.

All of the vector quantities in these equations are expressed in the world coordinate system. The inertia matrix $[{}_iI_i]$ in local coordinates is defined during system definition, and must be converted to world coordinates using the transformation*

$$[I_i] = [P_i]\,[{}_iI_i]\,[P_i]^T$$

The number of multiplications involved in this transformation is a primary justification for employing local coordinates in some dynamics formulations (see [Luh 1980]).

Note that the reaction force $\underline{f}_i$ in these equations is the force link i-1 must exert on link i for dynamic equilibrium. For the end-effector link N+1, the same equations are used except that the applied force $\underline{f}_p$ and torque $\underline{t}_p$ at the tool reference point replace the terms $\underline{f}_{i+1}$ and $\underline{t}_{i+1}$ (also, $\underline{R}_{N+2} = \underline{r}_p$).

Actuator drive torques. - Each joint of the manipulator provides a single degree of freedom of relative motion between the successive links it connects. This relative motion is actively driven by an actuator and drive mechanism. Therefore one component of the reaction force or torque at the joint must be supplied by the actuator system, while the remaining components are the bearing reactions. In addition to providing this joint reaction component, the actuator must also overcome joint friction and its own inertia.

The torque (or force+) τ_i that actuator i must supply to drive the system is given by

$$\tau_i = \underline{S}_i \cdot \underline{t}_i + I_{em_i}\ddot{\theta}_i + K_{ev_i}\dot{\theta}_i + K_{ec_i}\sigma(\dot{\theta}_i) \qquad \text{(Rotating Joint i)}$$

$$\tau_i = \underline{S}_i \cdot \underline{f}_i + I_{em_i}\ddot{\theta}_i + K_{ev_i}\dot{\theta}_i + K_{ec_i}\sigma(\dot{\theta}_i) \qquad \text{(Sliding Joint i)}$$

where the new terms in these equations are

I_{emi} - effective inertia of the actuator and drive system (e.g., gearing);

K_{evi} - effective viscous friction coefficient;

K_{eci} - effective coulomb friction coefficient.

The signum function is defined by $\sigma(\dot{\theta}) = \begin{cases} 1 & \dot{\theta} > 0 \\ -1 & \dot{\theta} < 0 \end{cases}$

*This relation is readily derived from the fact that the rotational kinetic energy, a scalar, is invariant with respect to coordinate reference, so

$${}_i\underline{\omega}_i^T\,[{}_iI_i]\,{}_i\underline{\omega}_i = \underline{\omega}_i^T\,[I_i]\,\underline{\omega}_i$$

+If joint i is a rotating joint, τ_i has units of torque, while for a sliding joint i, τ_i is a force. The term "torque" will be used in either case.

For $\dot{\theta} = 0$, $\sigma(\dot{\theta})$ can assume any value between -1 and 1; it is generally selected to be zero in this case.

All of the terms previously defined are effective values referenced to joint displacement coordinates to promote computational efficiency. Preprocessing may be necessary to convert the actual values to their effective values. As an example, suppose the actual inertia of the actuator rotor is I_{am} and that this motor drives the joint through a gear reduction ratio of n_a (i.e., $\theta_a = n_a \theta$). The effective inertia I_{em} of the actuator motor is then given by

$$I_{em} = n_a^2 I_{am}$$

The corresponding conversion for the other terms in the actuator torque equations are

$$K_{ev} = n_a^2 K_{av}$$

$$K_{ec} = n_a K_{ac}$$

$$\tau = n_a \tau_a$$

Presentation of results. - During execution of the requirements analysis, print and plot files containing the significant analysis results are generated. The print file is a formatted output file that can be viewed at a terminal or spooled to a high-speed line printer to provide a hardcopy listing of the results. The results saved in the plot file can be displayed on an interactive graphics terminal or plotted on a hardcopy plotter. This utility is described in the Postprocessing Tools section of this report.

In addition, the motion of the manipulator system can be displayed on an Evans and Sutherland graphics workstation during execution of the requirements analysis. A display file can also be saved for subsequent replay.

Response Simulation

Response simulation involves evaluation of the motion and force trajectories of the manipulator system when driven by a prescribed set of actuator torques or a specified control law and reference command. In the ROBSIM implementation, the dynamic equations of motion are solved for the joint accelerations at each processing timestep. A Runge-Kutta fourth-order integration algorithm is used to numerically integrate these accelerations to obtain the joint velocities and positions. To calculate the joint accelerations for a specified state (position and velocity) and driving torques, the equations of motion must be reformulated to explicitly represent these accelerations. The appropriate form for the controlling equations is

$$\underline{\tau} = [A(\underline{\theta})]\underline{\ddot{\theta}} + \underline{b}(\underline{\theta},\underline{\dot{\theta}})$$

where $\underline{\tau}$ is the vector of actuator driving torques, $[A(\theta)]$ is the effective inertia matrix referenced to joint coordinates, and $\underline{b}(\underline{\theta},\dot{\underline{\theta}})$ is a vector of position- and velocity-related effective torques. The calculation of these terms is described in the following subsections.

Actuator driving torques. - The actuator torques driving the response simulation can be generated by one of the following methods:

1) Read a file of actuator torques versus time;

2) Read a file of actuator voltages versus time;

3) Use a feedback control law.

The first of these methods can be used to determine response to specific torque command profiles, such as a sinewave input. Alternately, an actuator torque file generated during a requirements analysis run could subsequently be used to drive the response simulation, thereby providing validation that the requirements analysis and response simulation agree.

Figure 9 illustrates some results from a simulation run of this type. A set of joint driving torques was generated by the requirements analysis for a motion consisting of (1) a straight-line end-effector move for the first second, (2) a 0.5-second wait, (3) a 1-second return to the original position (joint-interpolated move), and (4) a hold in this position. Figure 9 shows the desired trajectory θ(req) for two intermediate joints of a six-degree-of-freedom arm, as well as the simulation results for these joints using two different integration timesteps, 0.01 second and 0.005 second. This figure shows how the simulation error tends to increase with time, and that very small timesteps are needed to accurately track desired trajectories for long motion segments.

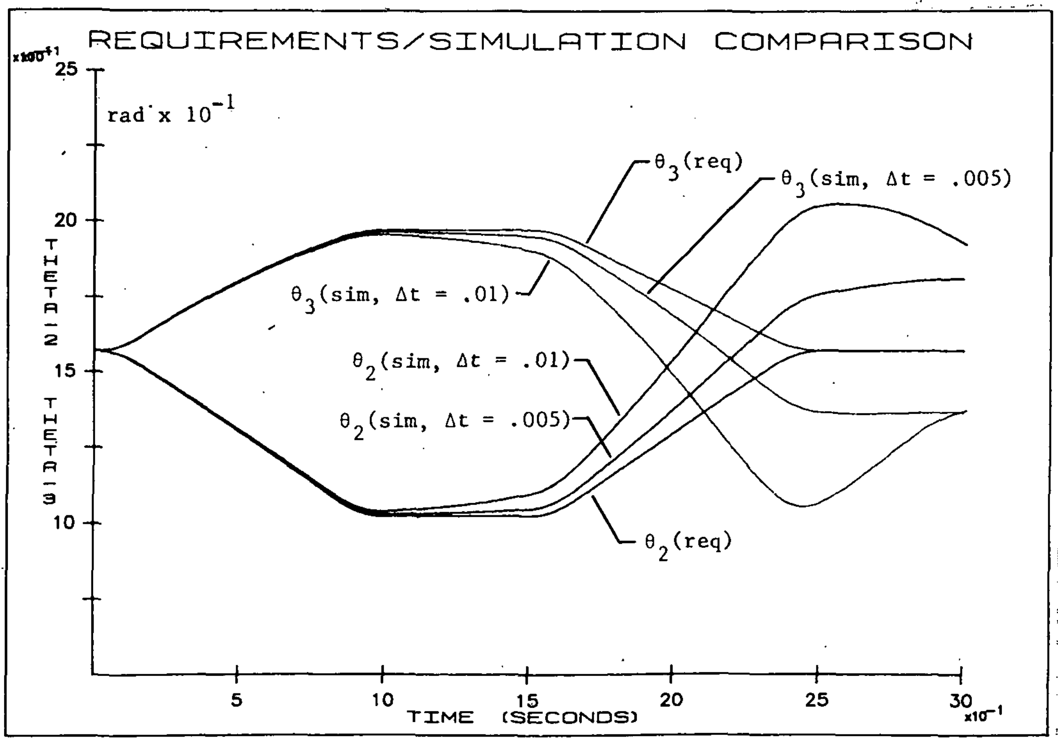


Figure 9.-
Results of simulation driven by torques generated in requirements analysis.

Actuation methods 2) and 3) are used with the dc torque-motor model implemented in ROBSIM. The torque supplied by motor i is proportional to the armature current i_i:

$$\tau_i = K_{et_i} i_i$$

where K_{eti} is the effective torque constant of the motor. The armature current is a function of the applied voltage V_i

$$V_i = L_i \frac{di_i}{dt} + R_{a_i} i_i + K_{emf_i} \dot{\theta}_i$$

where L_i is the armature inductance, R_{ai} is the armature resistance and K_{emfi} is the effective back-emf coefficient for the motor. In the motors modeled to date, the armature inductance has been found to be so small (hence, the electrical time constant is very short) that an unrealistically small integration timestep is required to track the electrical dynamics of the motor. Therefore, the inductance is neglected in the current motor model and the torque can be related to the applied voltage by combining the above equations to obtain

$$\tau_i = \frac{V_i - K_{emf_i} \dot{\theta}_i}{R_{a_i}} K_{et_i} \qquad (*)$$

In this equation, all coefficients are referenced to the joint displacements coordinates $\underline{\theta}$. If, for example, the gear reduction ratio is n_a, the effective values are related to the actual motor coefficients K_{at} and K_{aemf} by

$$K_{et} = n_a K_{at}$$

$$K_{emf} = n_a K_{aemf}$$

If a voltage file is read, equation (*) is then used to compute the actuator driving torque at each timestep. At intermediate times for which no voltage has been saved in the voltage file, linear interpolation is used to determine the voltage value.

Rather than reading a previously established file of command voltages, the motor drive voltages can be generated during the simulation run using a feedback control law. The implementation of feedback control models in the ROBSIM program is discussed in a subsequent section on manipulator controllers.

Position- and velocity-related torques. - The vector $\underline{b}(\underline{\theta},\dot{\underline{\theta}})$ of position- and velocity-related torques (see bottom of page 35) includes the effects of static loads (such as gravity and applied forces), friction, and the velocity-related inertia terms ("centripetal" and "Coriolis" torques). Because this effective torque contribution depends on the current manipulator state and includes all torque terms except the inertia torques attributable to joint accelerations, the vector $\underline{b}$ is conveniently and efficiently computed by performing the requirements analysis with the joint accelerations $\ddot{\underline{\theta}}$ set to zero. Therefore no additional program modules need to be implemented for the evaluation of this term.

Effective inertia matrix. - The matrix $[A(\underline{\theta})]$ of position-dependent inertia coefficients provides the relation between joint accelerations and actuator torques, including dynamic crosscoupling (i.e., acceleration at joint i produces reaction torques at the other joints). This matrix is symmetric and positive-definite; the kinetic energy of the manipulator in motion is given by the quadratic form

$$K.E. = \tfrac{1}{2}\dot{\underline{\theta}}^T \, [A(\underline{\theta})]\dot{\underline{\theta}}$$

An efficient algorithm for calculating the inertia matrix, based on the results in [Thomas 1982] and [Walker 1982], is implemented in the ROBSIM package. A brief presentation of the equations used is given below.

Consider link i of the manipulator with mass m_i, centroid location $\underline{r}_{cgi}$ and inertia distribution $[I_i]$ (in the current configuration). The effective inertia $[A^i]$ due to this link's mass distribution is given by

$$[A^i] = [J^i]^T \left[\begin{array}{c|c} [M_i] & 0 \\ \hline 0 & [I_i] \end{array}\right] [J^i]$$

and where $[M_i]$ is a 3x3 diagonal matrix with the link mass M_i along the diagonal, and where $[J^i]$ can be written

$$\underline{J}^i_j = \begin{Bmatrix} \underline{S}_j \times (\underline{r}_{cg_i} - \underline{R}_j) \\ \\ \underline{S}_j \end{Bmatrix} \qquad \text{(Rotating Joint } j \leq i)$$

$$\underline{J}^i_j = \begin{Bmatrix} \underline{S}_j \\ 0 \\ 0 \\ 0 \end{Bmatrix} \qquad \text{(Sliding Joint } j \leq i)$$

For $j > i$, the components of $\underline{J}^i_j$ are zero because displacement at joint j has no effect on the absolute motion of link i. The total inertia matrix is obtained by summing the contributions from each link i. The efficiency of the algorithm is further improved by using the symmetry of [A] to reduce the number of terms calculated. Also, composite masses, centroids and inertias consisting of the mass of all links from joint i to the free end of the arm are computed for each i. This allows each term of the expression for [A] to be evaluated for only one composite mass instead of for each individual link's mass (see [Walker 1982]).

The mass and inertia of a load being carried by the manipulator are included in the matrix of inertia coefficients by adding them to the mass distribution of the end-effector. In addition, the effective inertia, I_{emi}, of actuator i is added to the ith diagonal term of [A].

Because the effective inertia matrix is positive-definite, the equations of motion

$$\underline{\tau} = [A(\underline{\theta})]\underline{\ddot{\theta}} + \underline{b}(\underline{\theta},\underline{\dot{\theta}})$$

can be solved uniquely to obtain $\underline{\ddot{\theta}}$ in terms of the state $(\underline{\theta},\underline{\dot{\theta}})$ and driving torques $\underline{\tau}$.

<u>Joint friction</u>. - As discussed in the Requirements Analysis section, two types of joint friction--viscous and coulomb--are implemented. The viscous friction is readily included in the torque vector $\underline{b}(\underline{\theta},\underline{\dot{\theta}})$ as is the coulomb friction as long as the joint velocity is <u>not</u> zero. Special methods for handling the coulomb friction term must be employed when the velocity $\dot{\theta}_i$ at some joint is zero.

Figure 10 shows the joint coulomb friction τ_c as a function of joint velocity. This figure illustrates that when the joint velocity is zero, the coulomb friction torque value is not uniquely defined, but lies somewhere between the minimum and maximum values $-K_{es}$ and K_{es}, where K_{es} is the effective static friction coefficient for the joint. In the case when one or more joints have zero velocity, the set of equations to be solved for the joint accelerations can be written

$$[A(\underline{\theta})]\underline{\ddot{\theta}} = \underline{\tau} - \underline{b}(\underline{\theta},\underline{\dot{\theta}}) + \underline{\tau}_c$$

$$-K_{es_i} \leq \tau_{c_i} \leq K_{es_i}$$

$$\tau_{c_i} = \rho(\ddot{\theta}_i, K_{ec_i}, -K_{ec_i})$$

where $\underline{\tau}_c$ is the vector of joint friction torques for the zero-velocity joints, and the selection function ρ has the special definition

$$\rho(c,x^-,x^+) = \begin{cases} x^- & ; \; c<0 \\ x^+ & ; \; c>0 \\ x \text{ (arbitrary)} & ; \; c=0 \end{cases}$$

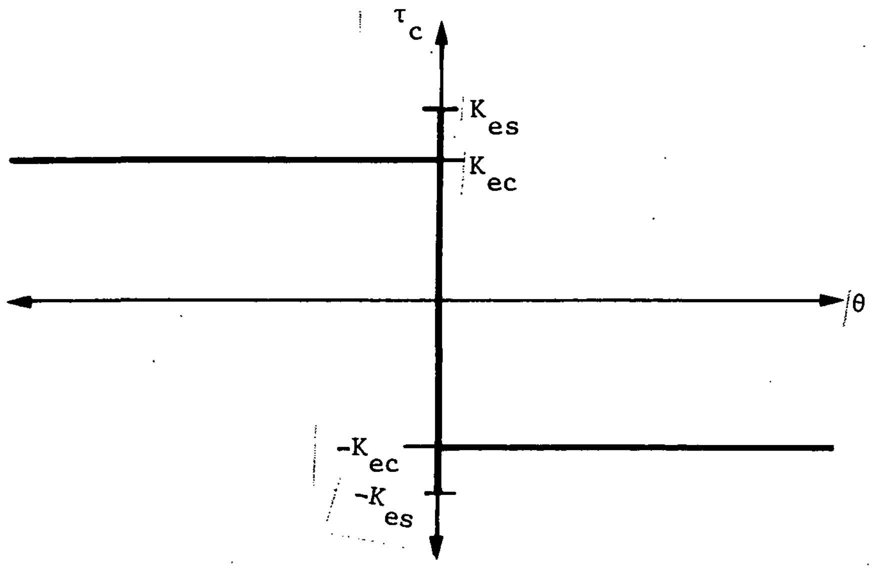


Figure 10.- Coulomb/static friction at joint.

A special solution procedure is implemented for solving equations of the type given previously; it is briefly described at the end of this section of the report. In the case represented here, a unique solution is obtained for $\underline{\theta}$ and $\underline{\tau}_c$.

The procedure previously described determines when relative motion at a joint begins. Another special procedure must be employed to evaluate when moving joints stop because of coulomb/static friction. In the ROBSIM implementation, whenever the value of the velocity at some joint switches sign between one integration step and the next, the joint velocity is reset to zero and the algorithm described above is used to determine whether the joint actually continues moving or comes to rest.

Figure 11 shows the effect of static friction on the motion of a single joint driven by a sinusoidal torque input. The response is dramatically affected by this nonlinear torque contribution.

Motion constraints. - The results considered so far assume that the joint displacements are independent, controlled variables and that there are no external constraints on the motion of the end-effector. Often during system operation, interactions with the environment constrain the motion of the tool (e.g., consider turning a crank or inserting a peg in a hole). Several methods exist for modeling these interactions in a computer simulation. For instance, if the external forcing functions are known, they can be included in the vector $\underline{b}$ as described in the subsection on requirements analysis.

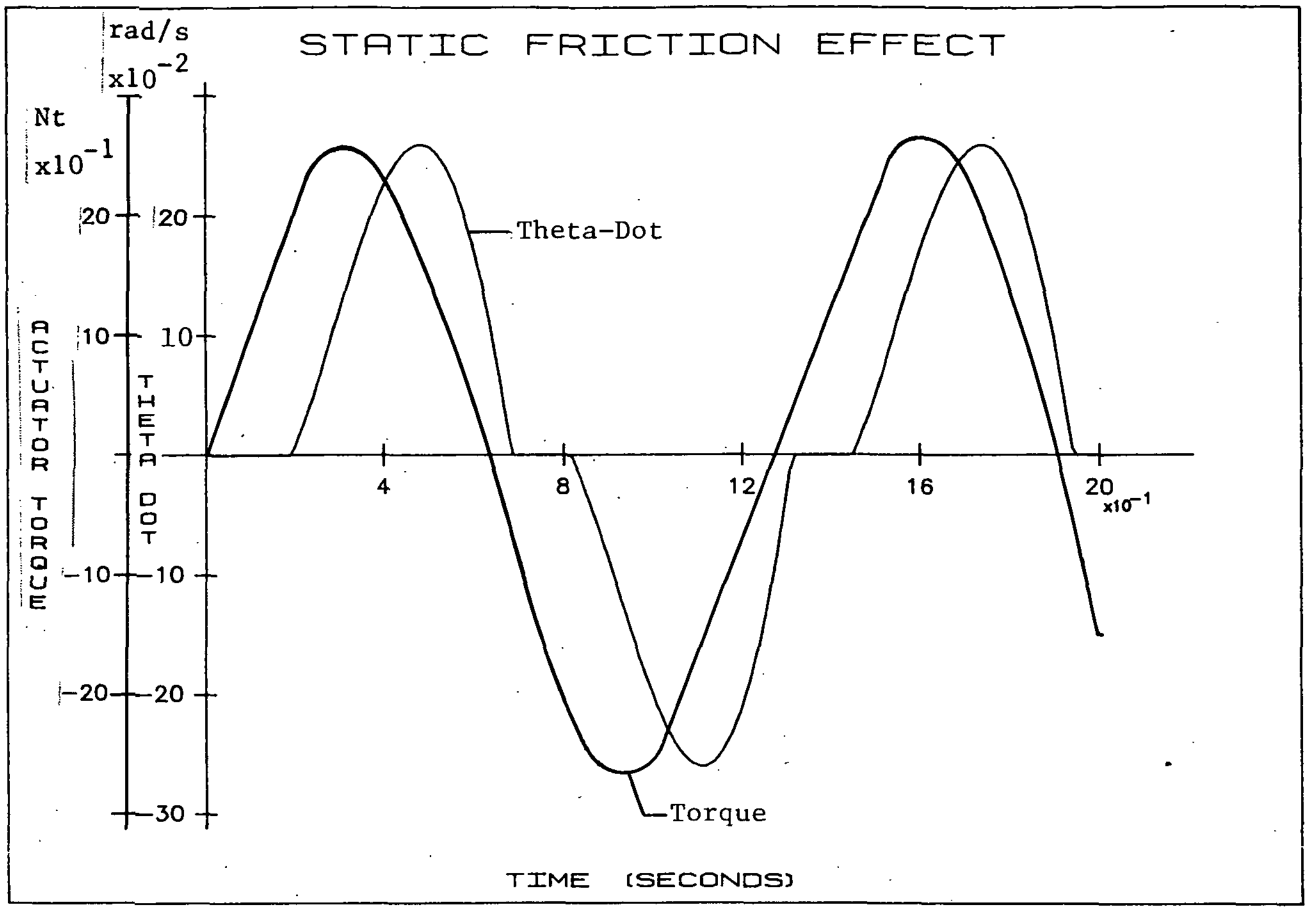


Figure 11.- Effect of static friction on system response.

One method for representing interactions with the environment is by a spring between the end-effector and some reference position in the inertial coordinate system. The ROBSIM dynamics package contains the capability for including a general compliance element between the end-effector and the fixed reference. In this implementation, the end-effector position and orientation are computed. Then a six-dimensional error vector $(\Delta \underline{r}_p^T, \Delta \underline{\phi}_p^T)^T$ is determined, which represents the translational and rotational error between the actual end-effector position and the spring reference position (in which the spring force would be zero). A position-dependent spring force $\underline{f}_p$ and a torque $\underline{t}_p$, given by

$$\begin{Bmatrix} \underline{f}_p \\ \underline{t}_p \end{Bmatrix} = [K_{sp}] \begin{Bmatrix} \Delta \underline{r}_p \\ \Delta \underline{\phi}_p \end{Bmatrix}$$

are applied to the end-effector. Here, $[K_{sp}]$ is a 6x6 stiffness matrix. These applied loads contribute to the reaction forces and torques at the joints and their effects are included in the vector $\underline{b}$ in the response simulation formulation.

While this spring formulation is readily understood and implemented, there are several drawbacks to using it for modeling motion constraints:

1) The user must specify a stiffness matrix that accurately characterizes the system interactions;

2) It is inconsistent with the rest of the manipulator model, which does not include compliance;

3) The stiffness matrix often contains very large values, thus requiring a restrictively small integration timestep to assure numerical stability.

Therefore, the ROBSIM simulation tool contains modules for incorporating rigid constraints on the motion of the end-effector. The basic form of the method implemented has previously been applied for the dynamic analysis of mechanisms (e.g., see [Chace 1971]), and solves for the constraint reaction forces as well as the joint accelerations. To include the effect of contact friction, only point constraints are considered. That is, constraints on the angular motion of the link cannot be specified directly, although such constraints can generally be represented by more than one point constraint.

Consider, for example, that the tip of the manipulator is pressing against the face of a rigid object as illustrated in Figure 12. The tip point r is constrained from further motion toward the obstacle. This constraint can be expressed as

$\underline{v}_r \cdot \underline{n}_r \geq 0$, where $\underline{v}_r$ is the velocity of the tip point r in contact with the obstacle and $\underline{n}_r$ is the outward pointing surface normal.

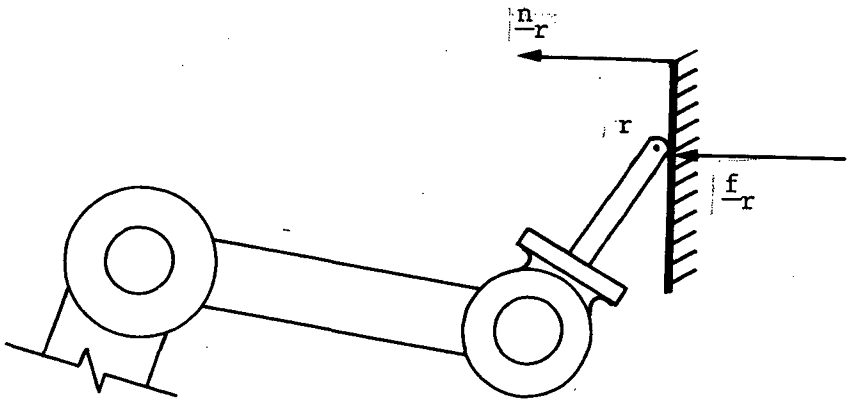


Figure 12.- Constrained motion of end-effector.

This is a single-acting constraint in that motion of the point in only one direction is restricted. For a double-acting constraint, the component of motion along $\underline{n}_r$ is restricted in both directions and the constraint relation becomes

$$\underline{v}_r \cdot \underline{n}_r = 0$$

This could result, for example, from a roller in a slide or a peg in a hole. If the obstacle is moving at a constant velocity, the constraint is handled identically, with the component of this velocity along $\underline{n}_r$ forming the right-hand side of these equations.

Assume that the velocity constraint is instantaneously satisfied exactly. To continue satisfying the constraint, the acceleration of point r is also restricted by the condition*

$$\underline{a}_r \cdot \underline{n}_r = 0$$

From the kinematic relations previously discussed, this can be transformed into the following constraint on the instantaneous joint accelerations:

$$\underline{w}_r^T \ddot{\underline{\theta}} + \underline{a}_r^v \cdot \underline{n}_r = 0$$

where

$$\underline{w}_r = [J_r]^T \underline{n}_r$$

and $[J_r]$ is the translational Jacobian for point r, and $\underline{a}_r^v$ is the velocity-related acceleration of point r (i.e., for the current joint velocity state with zero joint accelerations). The constraint on the motion of r is provided by a reaction force $\underline{f}_r$ applied through the point (see Fig. 12). This reaction force acts along a line parallel to the constraint direction

$$\underline{f}_r = f_r \underline{n}_r$$

but the magnitude f_r must be determined along with the joint accelerations. This applied force produces effective joint torques $\underline{t}_r$ given by (e.g., see [Thomas 1982])

$$\underline{\tau}_r = [J_r]^T \underline{f}_r = f_r \underline{w}_r$$

that must be included in the dynamic equations of motion.

Consider, for example, the case in which there is one double-acting constraint. The above relations lead to a set of N+1 linear equations that can be solved for the instantaneous accelerations $\ddot{\underline{\theta}}$ and reaction force magnitude f_r

$$\left[\begin{array}{c|c} [A] & -\underline{w}_r \\ \hline \underline{w}_r^T & 0 \end{array}\right] \left\{\begin{array}{c} \ddot{\underline{\theta}} \\ \hline -f_r \end{array}\right\} = \left\{\begin{array}{c} \underline{\tau} - \underline{b} \\ \hline -\underline{a}_r^v \cdot \underline{n}_r \end{array}\right\}$$

*For the single-acting constraint type, replace the equality sign by "greater than or equal to" in these relations.

Configurations containing single-acting constraints can be handled by employing the nonlinear, partial constraint function ρ defined previously in the subsection on joint friction. In this case, the equations to be solved for the joint accelerations and reaction force are:

$$[A]\ddot{\underline{\theta}} = \underline{\tau} - \underline{b} + f_r \underline{w}_r$$

$$f_r \geq 0$$

$$\underline{w}_r^T \ddot{\underline{\theta}} + \underline{a}_r^v \cdot \underline{n}_r \geq 0$$

$$f_r = \rho(\underline{w}_r^T \ddot{\underline{\theta}} + \underline{a}_r^v \cdot \underline{n}_r, -, 0)$$

The implicit function ρ is introduced to represent the fact that the reaction force must vanish as the point accelerates away from the constraint. (Note that the conditional cannot be less than zero in this case.)

The solution procedure described at the end of this section is again employed to solve these equations. In general, a unique solution will exist. However, if some joints have zero velocity and some constraints are active, the reaction force distribution may be indeterminate, although a unique set of joint accelerations will result. More than one motion constraint can be considered simultaneously by simply expanding the set of equations.

Impact of collision. - In the above equations, it was assumed that the point velocity already satisfied the motion constraint. However, when the point initially collides with the obstacle, this will not be true. In the actual case, this collision will introduce oscillatory transients in the motion. This rapid transfer of momentum can be modeled as an inelastic collision to determine the velocity of the system after impact. To evaluate the joint velocities after impact, consider the generalized momenta $\underline{Q}$, which are given by

$$\underline{Q} = [A]\dot{\underline{\theta}}$$

where [A] is the inertia matrix defined earlier. The change $\Delta\underline{Q}$ in generalized momenta must equal the generalized impulse $\underline{T}_I$ that results from the impulsive reactive force at the constraint. If the reaction impulse has magnitude f_I, then the generalized impulse will be

$$\underline{T}_I = [J_r]^T f_I \underline{n}_r = f_I \underline{w}_r$$

This leads to a set of linear equations for determining the change $\Delta\dot{\underline{\theta}}$ in the joint velocities during the impact:

$$\Delta\underline{Q} = \underline{T}_I \Longrightarrow [A]\, \Delta\dot{\underline{\theta}} = f_I \underline{w}_r$$

Combining these equations with the fact that the velocity constraint must be satisfied after impact leads to the following set of equations for obtaining the joint velocities after the collision

$$\left[\begin{array}{c|c} [A] & \underline{w}_r \\ \hline \underline{w}_r^T & 0 \end{array}\right] \left\{\begin{array}{c} \Delta\dot{\underline{\theta}} \\ \hline -f_I \end{array}\right\} = \left\{\begin{array}{c} 0 \\ \hline -\underline{v}_r \cdot \underline{n}_r \end{array}\right\}.$$

$$\dot{\underline{\theta}}_{after} = \dot{\underline{\theta}}_{before} + \Delta\dot{\underline{\theta}}$$

where $\underline{v}_r$ is the velocity of the constraint point before impact.

Coulomb friction at the constraint points. - In addition to the normal reaction force at the point of contact with the obstacle, there is also a friction force tangent to the restraining surface. This friction force, $\underline{f}_f$, is proportional to the normal force and directed opposite to the tangential velocity, i.e.,

$$\underline{f}_f = -\mu |f_r| \underline{e}_r$$

where μ is the coefficient of friction and $\underline{e}_r$ is a unit vector in the direction of tangential motion of the constrained point relative to the surface. The effective torque contribution at the joints is again found by premultiplying the friction force by the transpose of the Jacobian. For the single-acting constraint type, the equations are the same as given earlier, except the term $f_r\underline{w}_r$ is replaced by $f_r(\underline{w}_r - \mu\underline{w}_f)$ where

$$\underline{w}_f = [J_r]^T \underline{e}_r$$

In the case of a double-acting constraint, slightly more modification is needed because the magnitude of $\underline{f}_r$ can equal plus or minus f_r. This case is put into the standard form used above by replacing f_r with two variables f_r^+ and f_r^-, one of which is always zero depending on the sign of f_r. The joint torque contribution from the reaction normal force and friction force can be written

$$\underline{\tau}_r = f_r^+(\underline{w}_r - \mu\underline{w}_f) + f_r^-(\underline{w}_r + \mu\underline{w}_f)$$

The additional constraints for this case are

$$f_r^+ \geq 0$$

$$-f_r^- \geq 0$$

$$f_r^- = \rho(f_r^+, -, 0)$$

where the selection function ρ implies that $f_r^- = 0$ if $f_r^+ > 0$ and f_r^- has an arbitrary value (i.e., must be solved fom other constraints) when $f_r^+ = 0$.

As discussed for the joint friction case, the real difficulty in handling coulomb friction occurs at the singularity when the relative velocity between the interacting surfaces is zero. However, the ability to simulate this case is critical. For example, consider the insertion of a peg in a hole. It is vital to determine whether the insertion will proceed in the presence of coulomb friction, or whether the peg will instead jam in the hole. If jamming does occur, will the manipulator be able to free the peg and continue insertion?

To model the case of coulomb friction with zero tangential velocity, the friction force is approximated by two mutually perpendicular forces

$$\underline{f}_f = \mu(c_r^1 \underline{e}_r^1 + c_r^2 \underline{e}_r^2)$$

where $\underline{e}_r^1$ and $\underline{e}_r^2$ are two mutually perpendicular unit vectors that are also perpendicular to the surface normal (i.e., they define the instantaneous tangent plane). The coefficients c_r^i are bounded by the magnitude of the normal force

$$c_r^i + |f_r| \geq 0$$

$$-c_r^i + |f_r| \geq 0; \; i=1,2$$

Additionally, if the contact point has nonzero tangential acceleration, the friction force will have its limiting magnitude so

$$c_r^i = \rho(\underline{a}_r \cdot \underline{e}_r^i, \; |f_r|, \; -|f_r|) \; ; \; i=1,2$$

and

$$\underline{a}_r \cdot \underline{e}_r^i = (\underline{w}_f^i)^T \, \ddot{\underline{\theta}} + \underline{a}_r^v \cdot \underline{e}_r^i \; ; \; i=1,2$$

The magnitude of the normal force for the single- and double-acting constraint types for use in these relations is given by

$$|f_r| = f_r \text{ (single-acting)}$$

$$|f_r| = f_r^+ - f_r^- \text{ (double-acting)}$$

This method introduces two additional unknowns (c_r^i) and six additional constraints (four linear inequalities, two nonlinear partial constraints), all in the standard form developed previously. Some results using these equations are presented in a subsequent section of the report.

Modeling the ISRL Multisensor End-Effector

One component of today's manipulators that is woefully inadequate compared to the dexterity and sensitivity of its human counterpart is the manipulator end-effector or hand. Advanced mission applications will surely require more sensing capabilities than employed in industrial robots, which typically have at most a single presence detector, and more handling dexterity than the uncontrolled opening and closing of simple pneumatic grippers. This section describes the simulation of advanced end-effectors that incorporate several types of internal and external sensing and controlled gripping actions and discusses a specific implementation--the microprocessor-controlled gripper/sensor system developed for and in use at the Intelligent Systems Research Laboratory (ISRL) at NASA-LRC. The end-effector consists of a parallel jaw gripper mechanism with force sensors and proximity sensors mounted in the jaws and is attached to the manipulator through a six-degree-of-freedom wrist force sensor. The simulation model includes the geometry and dynamics of gripper operation, influences of load inertias on arm operation, wrist and jaw force sensors, and jaw-mounted proximity sensors.

<u>Configuration of the ISRL end-effector</u>. - Figure 13 displays the ISRL end-effector and Figure 14 illustrates the functional components of the gripper. The end-effector mechanical design and concept for proximity detection originated at the University of Rhode Island. The body of the end-effector houses a dc torque motor. A nylon gear on its shaft drives an incremental shaft encoder and a dc tachometer providing position and rate feedback; the gearing provides a 1:2.78 speed increase. The encoder output is wired directly into the microprocessor controller and the tachometer output is summed with the position error signal in the servoamplifier (Fig. 14). A limit sensor indicating the fully open configuration of the jaws provides a reference for the incremental position encoders. A worm gear on the motor shaft drives two opposing sector gears, each of which is attached to a link in a parallelogram mechanism that provides translation of the corresponding jaw, keeping the jaws parallel during opening and closing. The worm gear ratio is 100:1.

In addition to the gripping mechanism, motor and sensors, the ISRL end-effector incorporates proximity sensors in each jaw, a crossfire presence detector, multifreedom strain gage force sensors at each jaw mount, and a six-degree-of-freedom wrist force sensor. These sensors are described in subsequent sections. Also, the end-effector mechanism contains a spring-loaded mounting base with a limit sensor that provides overload protection when the device is in operation.

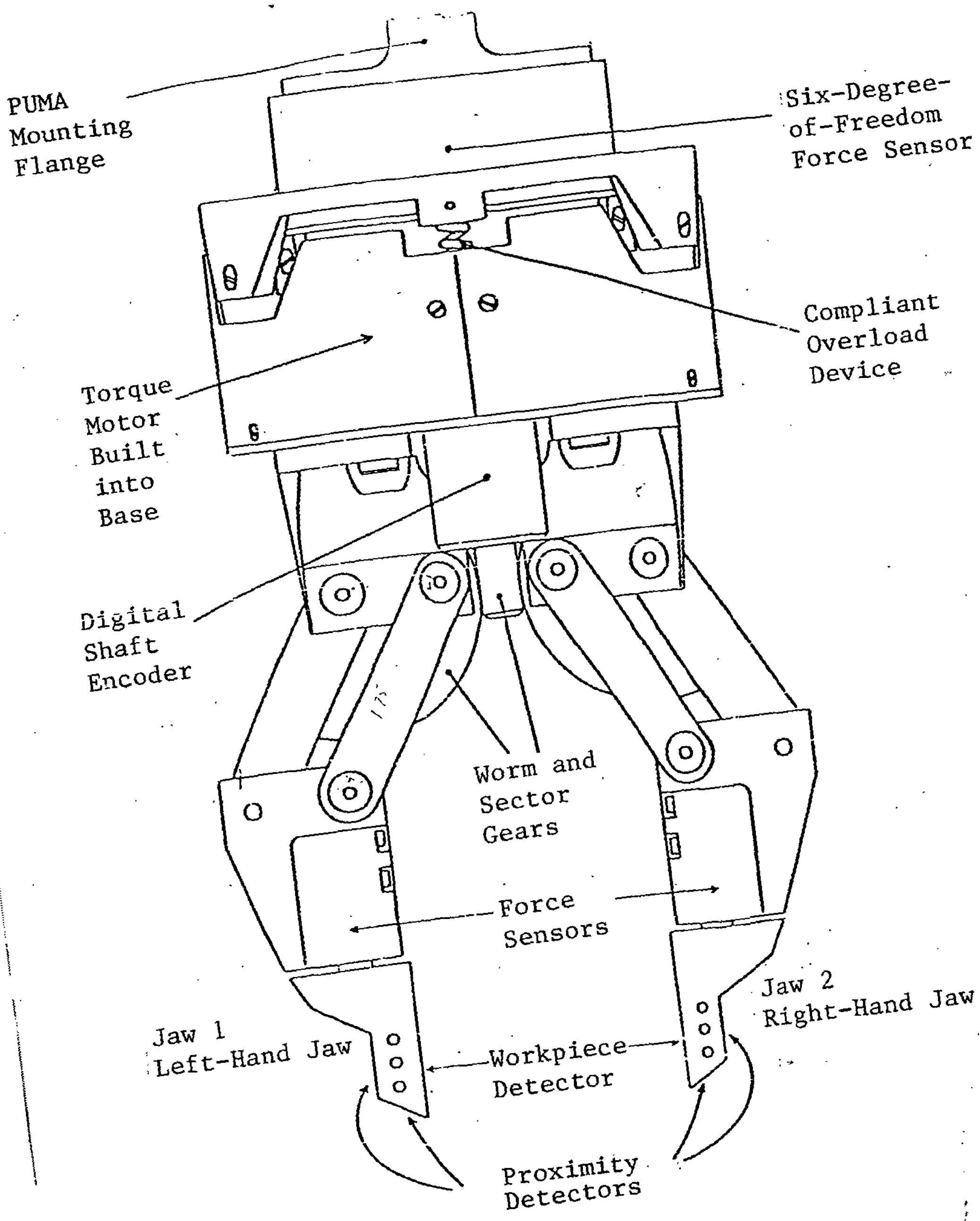


Figure 13.- ISRL end-effector.

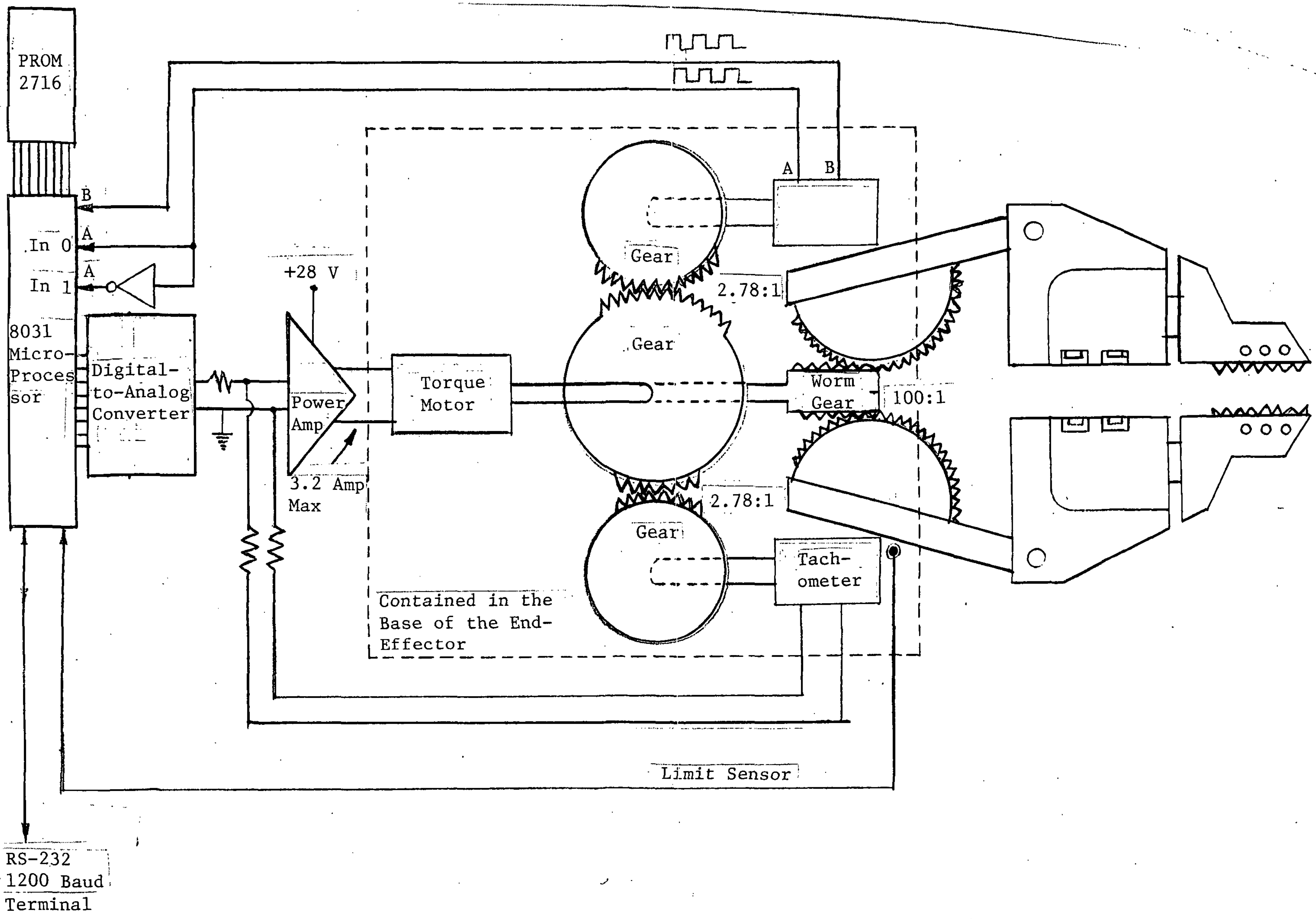


Figure 14.- Functional components of the ISRL gripper.

Geometry of gripper operation. - The ISRL end-effector contains one degree of freedom, specified by the reference angle $_{gr}$ shown in Figure 15. This is the angle between the reference z-axis and the centerline of the supporting gripper links. The motor angle, encoder angle and tachometer rate are related to the gripper angle by

1) Motor angle = $100\,\theta_{gr}$;

2) Encoder angle = $278\,\theta_{gr}$;

3) Tachometer rate = $278\,\dot{\theta}_{gr}$.

The end-effector reference system is chosen with the z-axis directed outward from the gripper along the motor axis, and the y-axis in the plane of motion of the jaws 9 (Fig. 15). The origin of this coordinate system is chosen as the midpoint of the wrist force sensor so the reaction force computed for this point can be used for modeling the wrist sensor results. Local reference systems are associated with each jaw of the gripper for defining the motion of such jaw-mounted components as the proximity sensors. Because the jaws do not rotate relative to the gripper body, their coordinate system remain parallel to the end-effector reference. The vectors $_{N+1}\underline{h}_{LJ}$ and $_{N+1}\underline{h}_{RJ}$ define the instantaneous location of the jaw coordinate systems relative to the end-effector reference and are given by the relations

$$_{N+1}\underline{h}_{LJ} = \underline{h}^{ref}_{LJ} + \ell \begin{bmatrix} 0 \\ -\sin\theta_{gr} \\ \cos\theta_{gr} - 1 \end{bmatrix}$$

$$_{N+1}\underline{h}_{RJ} = \underline{h}^{ref}_{RF} + \ell \begin{bmatrix} 0 \\ \sin\theta_{gr} \\ \cos\theta_{gr} - 1 \end{bmatrix}$$

where $\underline{h}^{ref}_{LJ}$ is the vector in end-effector coordinates to the tip of the left jaw when the gripper angle θ_{gr} is equal to zero, etc and ℓ is the length of the connecting link (Fig. 15). The distance d_{gr} between the gripper jaws can be expressed in terms of the gripper angle as

$$d_{gr} = d^{ref}_{gr} + 2\ell \sin\theta_{gr}$$

where d^{ref}_{gr} is this distance in the reference position ($\theta_{gr} = 0$). The positions of the gripper jaws in terms of world coordinates are given by

$$\underline{R}_{LJ} = \underline{R}_{N+1} + [P_{N+1}]\,{}_{N+1}\underline{h}_{LJ}$$

$$\underline{R}_{RF} = \underline{R}_{N+1} + [P_{N+1}]\,{}_{N+1}\underline{h}_{RJ}$$

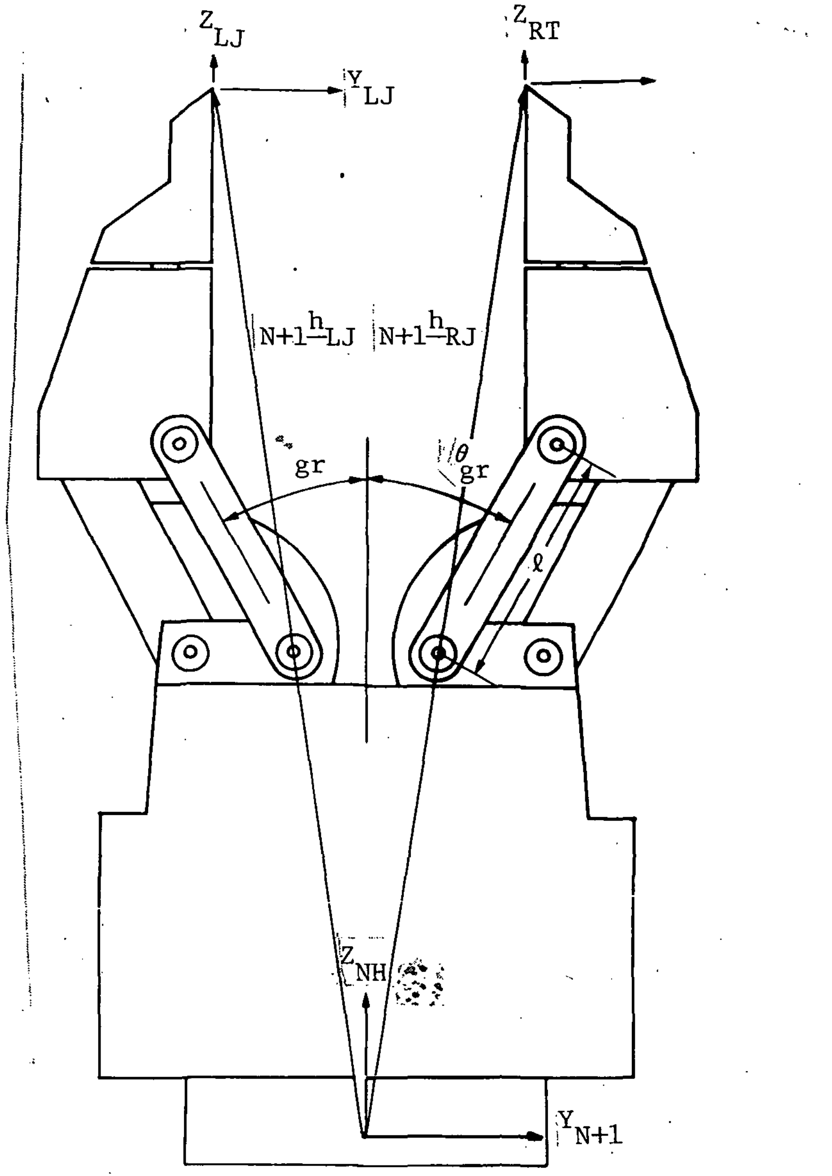


Figure 15.- Kinematics of the ISRL gripper.

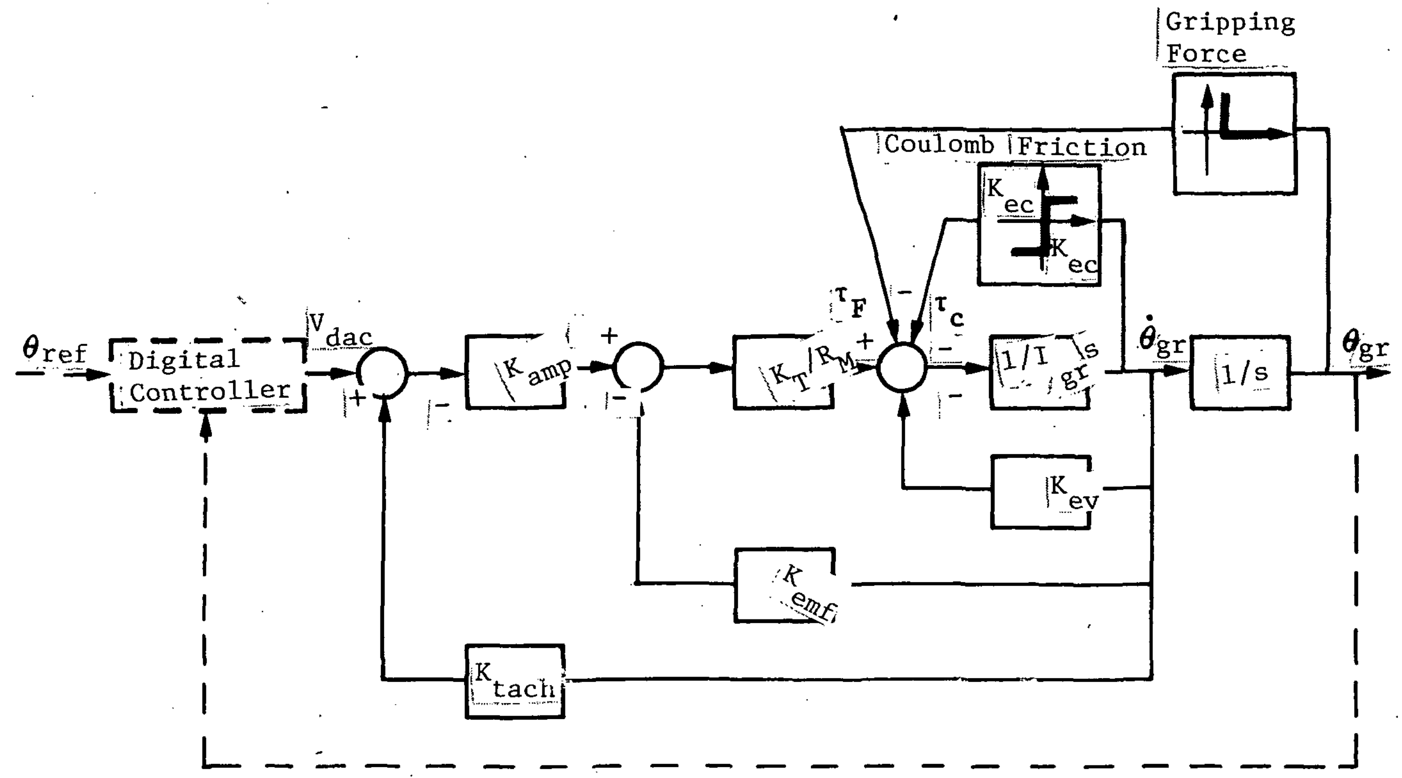
Then, any point whose position $_{LJ}\underline{r}$ in jaw coordinates is known (e.g., a proximity sensor fixed in the jaw) and can be located in world coordinates by the standard point transformation.

$$\underline{r} = \underline{R}_{LJ} + [P_{N+1}](_{LJ}\underline{r})$$

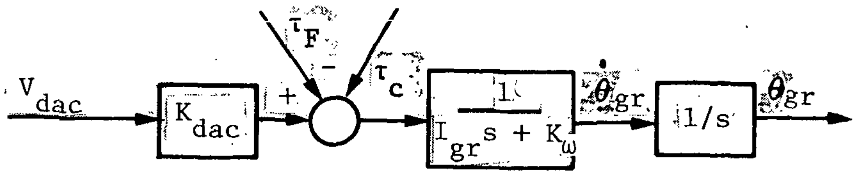
Gripper dynamics. - The dynamics associated with the single-degree-of-freedom gripper operation can be characterized by a second-order piecewise linear system. Figure 16(a) shows a block diagram representation of this system; it includes analog rate feedback from the tachometer, viscous friction, a coulomb friction term and the jaw gripping force. The symbols in this diagram include:

V_{dac}	-	output voltage from the digital-to-analog converter
K_{amp}	-	servoamplifier gain
K_{emf}	-	back emf coefficient for the gripper motor
K_{tach}	-	tachometer coefficient relating tach voltage to gripper speed
K_{ev}	-	effective viscous friction coefficient
K_{ec}	-	effective coulomb friction coefficient
K_T	-	effective torque constant for the motor
R_m	-	motor circuit resistance
I_{gr}	-	effective inertia of the gripper motor, gears, linkage and jaws
F_{gr}	-	gripping force
τ_F	-	effective torque due to gripping force
s	-	Laplace variable

The gripper angle θ_{gr} forms the reference coordinate for most of these terms (Fig.16); for example K_{ev} gives the torque referenced to θ_{gr} corresponding to velocity $\dot{\theta}_{gr}$. The voltage from the digital-to-analog converter is determined by the control law implemented in the microprocessor controller, gripper angle feedback from the digital encoder and a reference angle supplied by a user or host computer.



(a) Explicit form.



(b) Simplified form.

Figure 16.- Block diagram representing the dynamics of gripper operation.

The terms in Figure 16(a) can be combined to form the simplified block diagram shown in Figure 16(b). The new coefficients in this diagram are defined in

$$K_{dac} = \frac{K_T}{R_m} K_{amp}$$

$$K_\omega = \frac{K_T}{R_m} (K_{Tach} K_{amp} + K_{emf}) + K_{ev}$$

The digital-to-analog converter acts as a zero-order hold, so v_{dac} is a piecewise constant voltage. Assuming that the grippers is not contacting a workpiece and that the gripper velocity does not change sign (so that τ_c is a constant), the dynamic equation represented by Figure 16(b) can readily be solved to obtain the time trajectory of θ_{gr} during each period of constant voltage. The controlling equation is

and the solution can be written

$$\dot{\theta}(t) = \frac{K_{dac} V_{dac} - \tau_c}{K_\omega} (1 - e^{-\zeta(t-t_o)}) + \dot{\theta}_o e^{-\zeta(t-t_o)}$$

$$\theta(t) = \theta_o + \frac{K_{dac} V_{dac} - \tau_c}{K_\omega} [(t-t_o) + \frac{1}{\zeta} (e^{-\zeta(t-t_o)} - 1)]$$

$$-\dot{\theta}_o \frac{1}{\zeta} (e^{-\zeta(t-t_o)} - 1)$$

where t_0 is the time at the beginning of the segment, θ_0 and $\dot{\theta}_0$ are the position and velocity at the t_0 and

$$\zeta = \frac{K_\omega}{I_{gr}}$$

In a computer simulation, these equations can be used to evaluate the transition in the gripper state during one timestep of its controller unless the gripper velocity changes sign or the gripper closes on a workpiece during this transition. In either of these cases, the driving torque (right-hand side of the differential) changes during the timestep.

If the gripper jaw motion reverses direction, the time at which this occurs can be obtained from the previous velocity equation. The gripper position at that instant is determined and then the transition continues with the new values for the coulomb friction term, t_0, θ_0 and $\dot{\theta}_0$. When the gripper closes on an object, the same procedure can be employed, but the differential equation is changed to include a torque term proportional to the distance between the gripper jaws to simulate the compliance in the jaws and object. A simpler approach that can be used to evaluate the steady-state gripper force corresponding to a given applied voltage involves modeling the gripper impact as an inelastic collision between rigid bodies. The effective torque τ_F due to the gripping force F_{gr} is given by

$$\tau_F = K_{dac} V_{dac}$$

The gripping force is found from

$$\tau_F = F_{gr} G_{gr}$$

where G_{gr} is the first geometric derivative relating the closing of the jaws to the gripper reference angle and can be obtained by differentiating the position relation, leading to

$$G_{gr} = 2\ell\cos\theta_{gr}$$

Grasping a load object. - Grasping an object can lead to constraints on the arm motion (if the object is constrained) or additional mass to transport during the motion. Suppose a load is rigidly grasped by the end-effector and is moved by the arm. The path of the load object is found by evaluating a relative translation ${}_{N+1}\underline{h}_\ell$ and rotation $[{}_{N+1}P_\ell]$ between the end-effector reference and the load object's coordinate system at the time of the grasp

$${}_{N+1}\underline{h}_\ell = [P_{N+1}]^T (\underline{R}_\ell - \underline{R}_{N+1})$$

$$[{}_{N+1}P_\ell] = [P_{N+1}]^T [P_\ell]$$

Then, after evaluating the end-effector position for each subsequent arm configuration, the load position can be determined using these relative positions by rewriting the equations as

$$\underline{R}_\ell = \underline{R}_{N+1} + [P_{N+1}]\,{}_{N+1}\underline{h}_\ell$$

$$[P_\ell] = [P_{N+1}]\,[{}_{N+1}P_\ell]$$

The dynamic effects of the load mass on the arm motions are modeled by adding the mass distribution of the load to that of the end-effector. The position of the load relative to the end-effector is used in the procedure for combining the rigid-body masses described earlier in the section that discusses the system definition function.

The simplest implementation of the grasp operation performs these steps regardless of whether the object was in position to be gripped and the grasp was successful. To determine the success or failure of the grasp, knowledge of the surface geometry and positions of the object and gripper jaws must be used. For example, consider a cylindrical peg being grasped by flat fingers (Fig. 17). The position of the peg is determined by the vector $\underline{r}_p$ to the center of one end of the peg and the direction cosine vector $\underline{S}_p$ along the peg axis (Fig. 17). The projection of the peg axis onto the plane containing the jaw surface is readily evaluated by transforming the coordinates of this axis into the local jaw reference system (Fig. 18). The critical points c_1 and c_2 that define the endpoints of this projection onto the jaw surface can then be found. If either end of the projection of the peg axis lies within the jaw surface, this is one critical point. Other critical points occur at the intersection of the peg axis projection with the edges of the jaw surface (Fig. 18). These critical points must be determined to evaluate the load distribution between the two gripper jaws. If there is no intersection between the peg axis projection and the jaw surface, the grasp fails and corrective action must be taken.

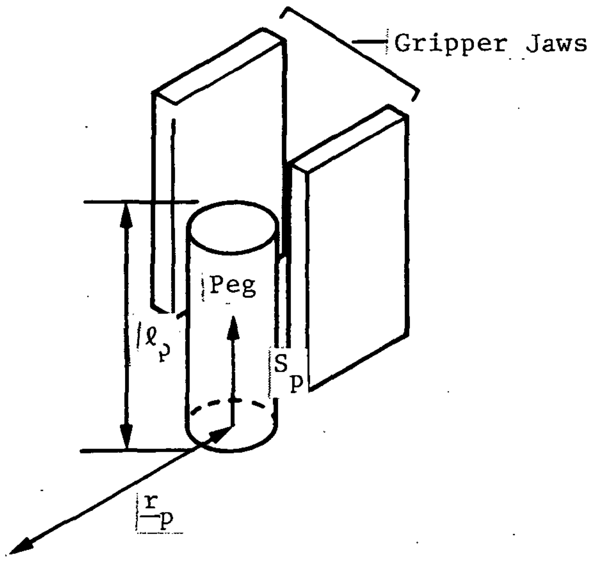


Figure 17.- Location of cylindrical peg.

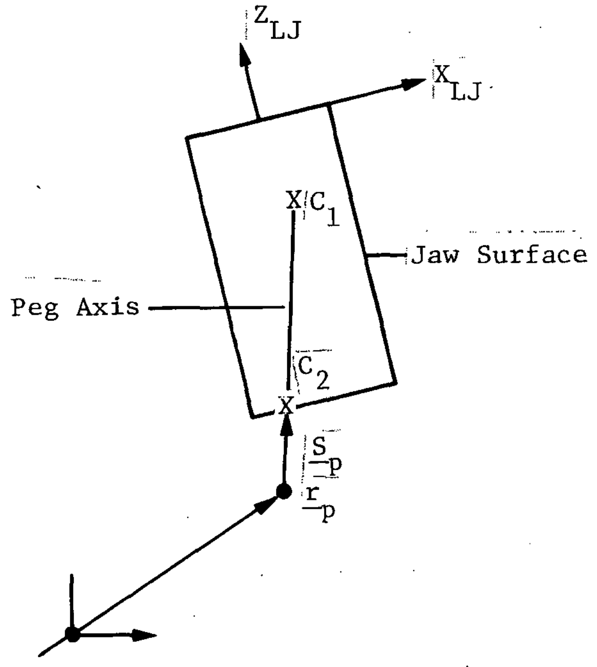
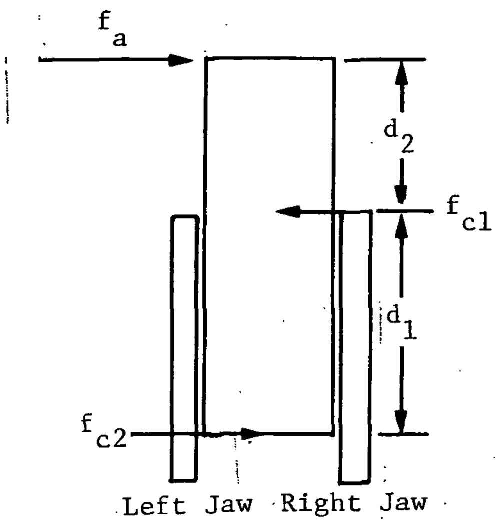


Figure 18.-
Critical contact points between peg and jaw surface.

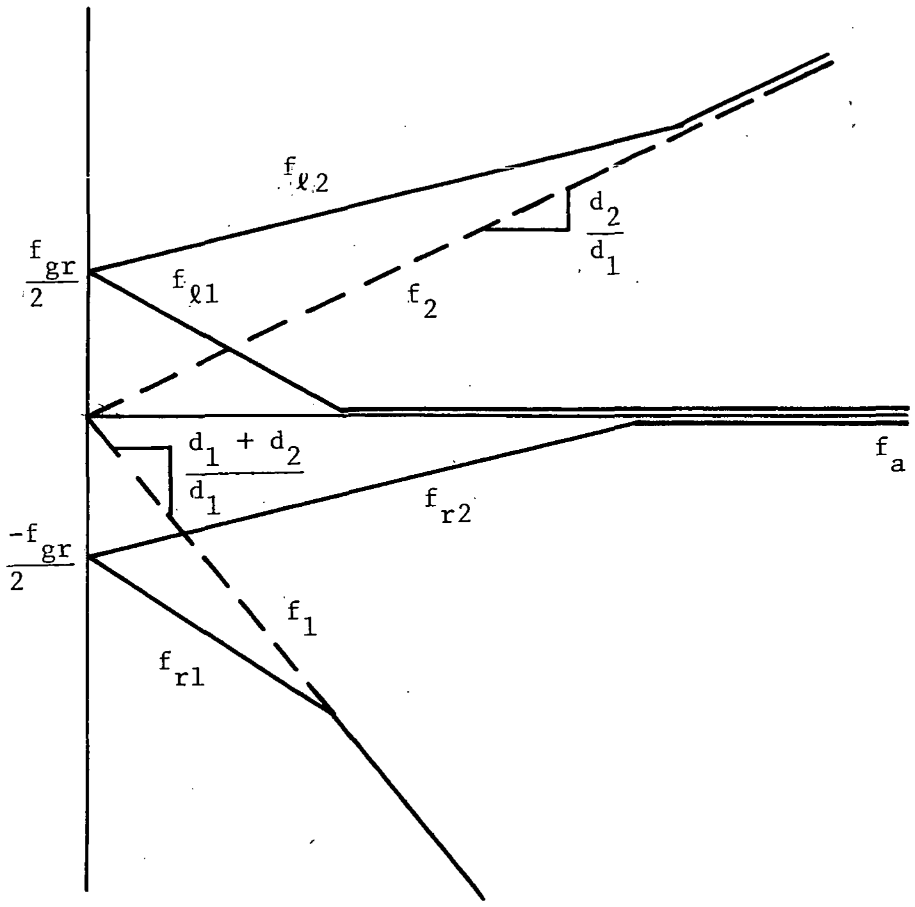
End-effector force sensing. - The ISRL end-effector is mounted to the arm through a six-degree-of-freedom force/torque wrist. By placing the reference point for the end-effector coordinate system at the center of this sensor, the force and torque computed during analysis for this reference point are the force and torque measured by the six-axis sensor.

In addition to the wrist sensor, the ISRL end-effector incorporates force sensors in each jaw of the gripper. These force sensors measure forces and torques in the local x- and y-coordinate directions of each gripper jaw (see Fig. 15). The distribution of applied loads between these two jaws depends on the critical contact points previously identified. The applied load on the peg is balanced by reaction force $\underline{f}_{C1}$ and $\underline{f}_{C2}$ at the critical contact points C_1 and C_2. The components of these forces in the local z-direction are neglected because the jaw force sensors have no sensing capabilities along or about this axis. The components in the local x-direction are assumed to be evenly distributed between the left and right jaws, and the y-axis force at each point acts on only one jaw depending on its direction. The y-component of the reaction forces at the jaw contact points also include terms for the gripping force applied. For example, consider a force of magnitude f_a applied to the peg as shown in Figure 19(a). This leads to reaction forces f_C, supplied by the right jaw, and F_{C2}, supplied by the left jaw. In addition, a gripping force of magnitude f_{gr} is applied. Figure 19(b) shows how these reaction forces are distributed between the left jaw (f_{l1} and f_{l2}) and right jaw (f_{r1} and f_{r2}) as the applied force magnitude varies. The reason that reaction forces greater than the gripping force can be supported is that the gripper worm gear is not backdriveable.

The jaw reaction forces generate forces and moments at the jaw force sensors. These forces and moments can be used during arm operation to identify the critical contact points and the external loads applied to the peg, although forces along, and moments about, the local z-axis cannot be identified, nor can the local x-coordinates of the critical contact points.



(a) Applied and reaction forces.



(b) Resulting jaw forces

Figure 19.-
Jaw forces resulting from an applied load and gripping force.

POSTPROCESSING CAPABILITY

Introduction

This section describes the capabilities implemented in ROBSIM for processing data generated during execution of the analysis tools functions. Because enormous amounts of data may be generated, methods other than viewing printouts are necessary to evaluate these data and compare different analysis runs. The following options are included in ROBSIM to accomplish this:

1) Graphics replay of simulation motion;

2) Graphics replay of simulation vs hardware motion;

3) Plots of manipulator parameters vs time.

Items 1) and 2) use an Evans and Sutherland picture system to display arm motions. The replay of simulation motion displays the manipulator system motions as they occurred during a requirements analysis or response simulation run. Replay of simulation vs hardware allows the user to display the motion of actual hardware (a "real" arm) and the motion of its software model simultaneously. The simulation model is displayed in the configuration defined during system creation. The hardware arm is shown as a stick figure superimposed on the simulation model. It should be noted that no calculations or analyses are carried out during either simulation replay or simulation vs hardware replay other than those needed for the Evans and Sutherland display.

Manipulator parameter plots allow the user to create x-y plots of various manipulator parameters as functions of time. Some of the parameters available for plotting are joint and end-effector positions, velocities, accelerations, forces and torques. These plots may be displayed on a graphics terminal or drawn on a pen plotter.

Simulation Replay

Replay of a manipulator system simulation displays the motions of the system exactly as they occur during a requirements analysis or response simulation run. The system is depicted as defined during system creation/modification with simple cylinder components or detailed geometries. Simulation playback makes use of a data file, written during the analysis function, that contains manipulator joint displacements as functions of time. These, and data defining the system geometry, are used to locate the positions and orientations of all joint/link pairs and load objects held by an arm with respect to the world coordinate system. The methods used to find these positions and orientations were described in the System Definition and Analysis Capabilities sections.

Because very few calculations are needed for the playback compared to the actual analysis, the motion is more continuous. As the playback proceeds, the motion may be halted at any time by pressing the button labeled zero on the Evans and Sutherland control panel. The display of the manipulator system may then be moved or rotated using the Evans and Sutherland control dials. This allows the user to view the system from a different perspective. Pressing the zero button again resumes the playback motion.

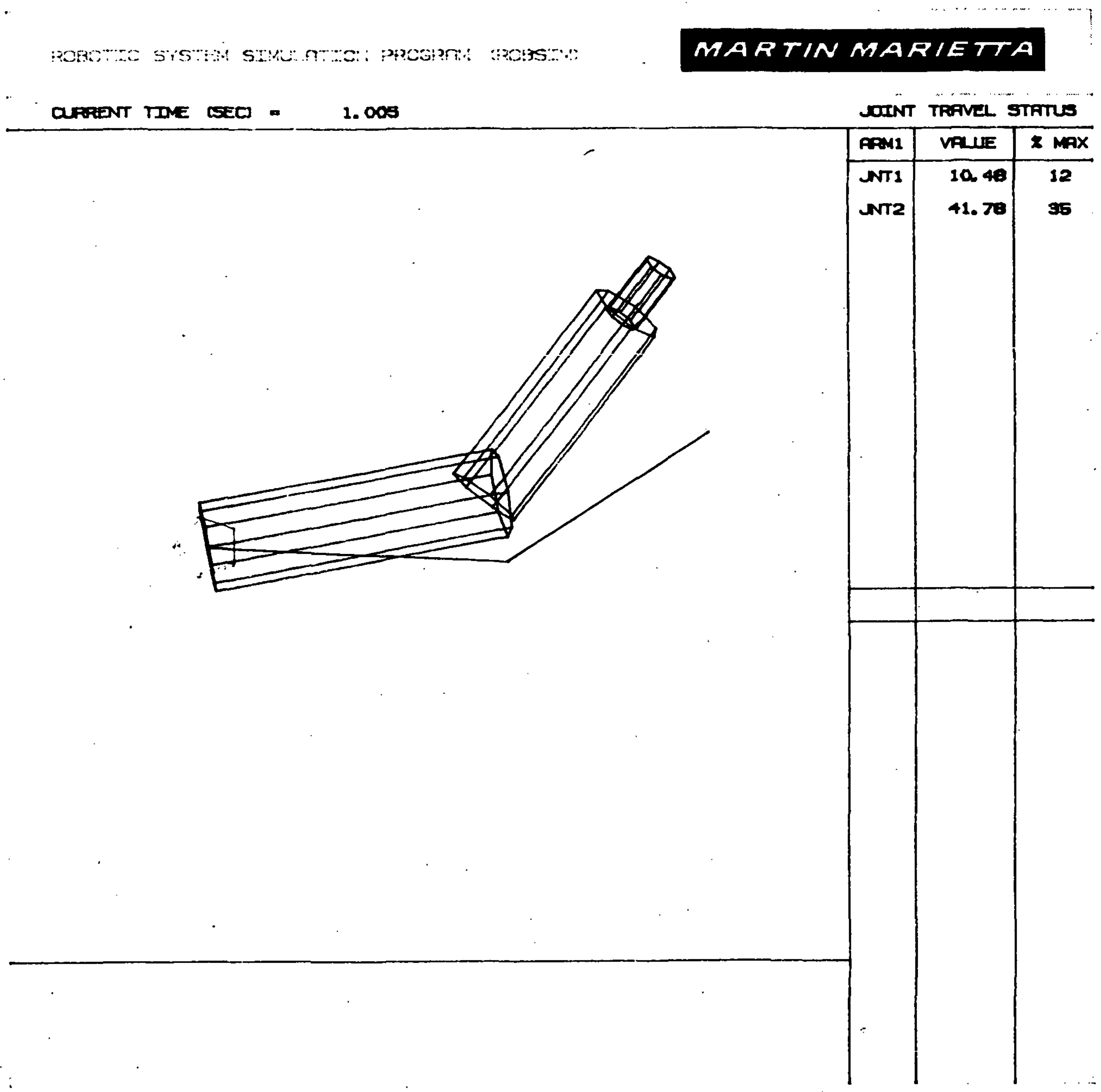
Simulation vs Hardware Replay

This postprocessing option allows the user to display the motion of a simulation as previously described, and concurrently display the motion of a hardware arm superimposed on the simulation. This is quite useful when validating system models. The data collected from a run of a hardware arm consist of joint actuator voltages and joint positions as functions of time, stored in a file, and transferred to the mainframe computer. A response simulation run modeling the hardware arm may use the hardware joint actuator voltages as control inputs or the same control algorithm as used for the hardware. This makes joint displacements as functions of time for both the hardware arm and its simulation model available for this option of the postprocessor. The joint displacements are used to find positions and orientations of the system components by the methods described earlier. The Evans and Sutherland picture system then uses the position and orientation data to display the motion of the two arms. The model, or simulation arm, is displayed as depicted during system definition; the hardware arm is displayed as a series of straight lines representing the centerlines of the arm links. As with the simulation replay described earlier, the motion of the display may be stopped at any time by pressing the zero button on the Evans and Sutherland control panel. The motion is suspended and the display may be viewed from a different perspective. Motion is resumed by pressing the zero button again. Figure 20 shows an Evans and Sutherland display of a planar arm simulation model and the stick figure representing the hardware arm.

Parameter Plots

This postprocessing option allows the user to plot various arm parameters as functions of time. A commercially available graphics software package is used to display the plot data on either a graphics terminal or a pen plotter. The choice of parameters available for plotting depends on the plot package chosen during the requirements analysis or response simulation run. The plot packages available for each type of analysis are tabulated.

Requirements analysis	Response simulation
1) Brief 2) End-effector data 3) Joint positions 4) Reaction forces 5) All or the above	1) Brief 2) End-effector data 3) Joint positions 4) Reaction forces 5) All or the above 6) PID control data 7) Force/torque control data



```
ROBOTIC SYSTEM SIMULATION PROGRAM (ROBSIM)          MARTIN MARIETTA

CURRENT TIME (SEC) =      1.005                     JOINT TRAVEL STATUS
                                                    ARM1   VALUE   % MAX
                                                    JNT1   10.48    12
                                                    JNT2   41.78    35
```

Figure 20. - Simulation vs hardware replay.

Listings of the manipulator variables saved by each plot package may be found in the analysis tools and postprocessor user's guides. Data pertinent to the chosen package are written to a file during the analysis run. During postprocessing, the user may plot any of the parameters saved in this file. The user determines the plot title, x- and y-axis labels and the number of curves per plot (up to 31). Figure 21 shows the displacements of joints 1 and 2 for a planar arm during response simulation using PID control.

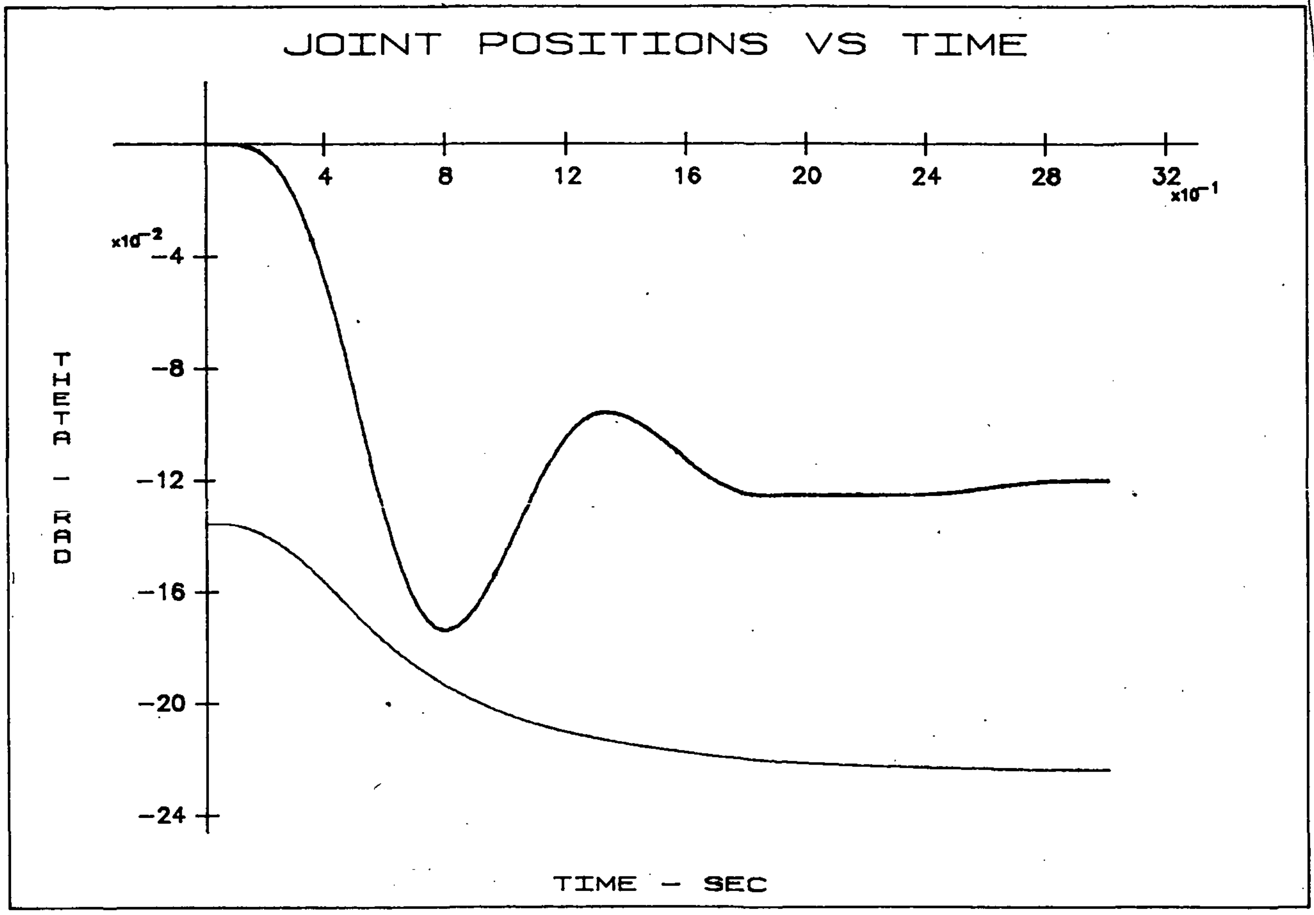


Figure 21. - Joint displacements.

MANIPULATOR CONTROL

As the number of manipulators used in various applications increases, the demands placed on control systems and sensor performance also increase. The areas in which the demands occur are accuracy (implying repeatability), speed, and uniformity of performance over a range of loading conditions. Efficiency of operation--implying the use of optimization techniques--is also becoming important, especially with regard to long-duration space missions and high-volume industrial operations. This section focuses on the different families of algorithms used for manipulator control.

In the Manipulator Description section the basic manipulator problem is defined in terms of fundamental elements--dyanmics, reference frame transformations, and performance requirements. The Control Algorithms section describes several conventional position control approaches. The control algorithms discussed are:

1) Linear discrete with feedforward nonlinear compensation;

2) Resolved acceleration;

3) Resolved motion force.

The Adaptive Control section discussion includes both linear adaptive control and nonlinear adaptive control. Simulation results are presented for both cases.

The Force/Torque Control section introduces the subject of control involving both positional and force constraints. Two approaches, hybrid control and active stiffness control, are analyzed in detail and simulation results presented.

Manipulator Description

A full description of the plant to be controlled is required as the first step in any control analysis. This section describes the properties of the manipulator important to control system synthesis:

1) Kinematics;

2) Dynamics;

3) Performance requirements.

Figure 22 shows a typical manipulator system.

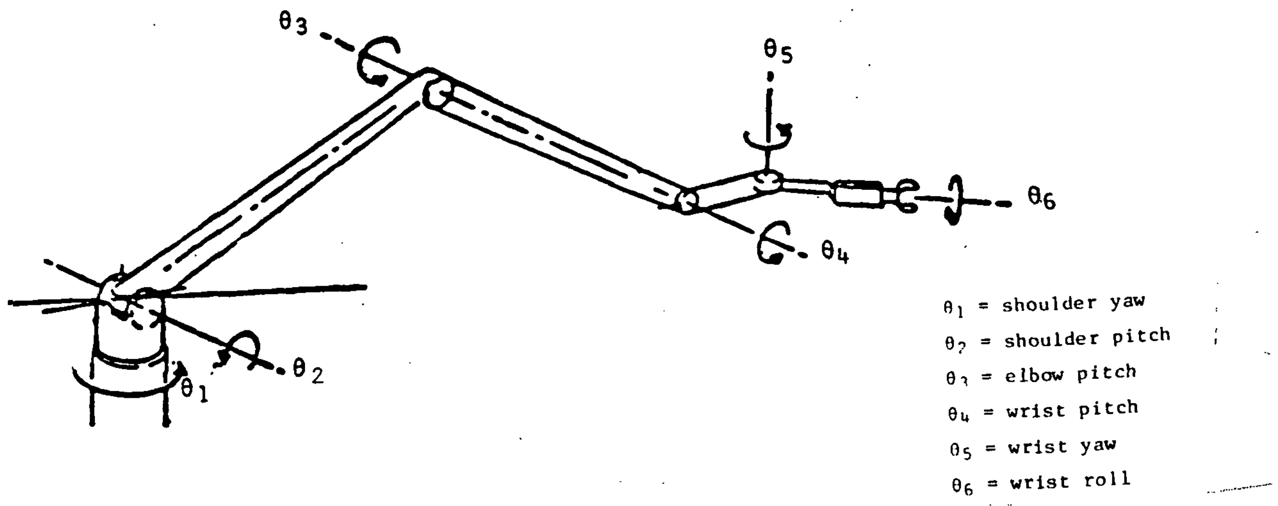


Figure 22. - Six-degree-of-freedom manipulator.

Kinematics. - The first area of concern, kinematics, involves transformations between the manipulator reference frame, the joint angles and linear displacements (in the case of sliding joints), and other Cartesian reference frames (work coordinates). Although the transformation equations are not difficult to determine conceptually, the function involves a significant amount of computation. After this, the problem then becomes one of determining a torque input vector $\underline{T}$ so actual joint positions will match desired joint positions with specified time and overshoot constraints (Fig. 23).

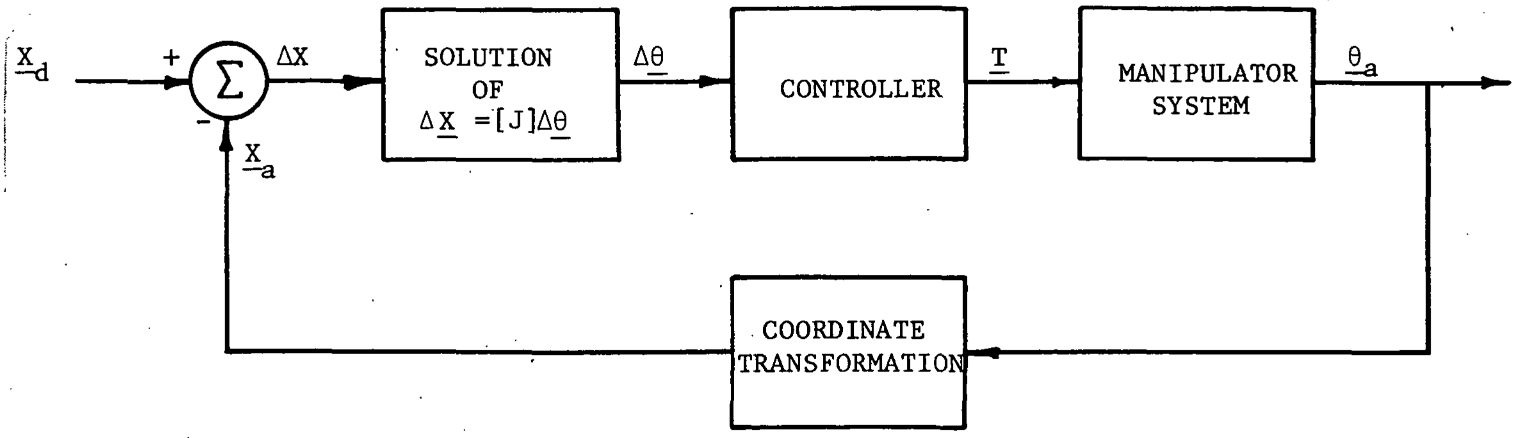


Figure 23. - Manipulator joint position solution procedure.

where

$\underline{X}_d$ is the desired end-effector position (in Cartesian coordinates)

$\underline{X}_a$ is the actual position of the manipulator (in Cartesian coordinates)

$\underline{\theta}_a$ is the vector of actual manipulator joint angles

$\underline{T}$ is the vector of torque inputs to be applied to the manipulator joints

This is the typical servomechanism control problem. The direct application of standard linear control theory is not possible, however, because of the nonlinearities and coupling inherent in the dynamics of the manipulator. A following section describes the basic methods employed for constructing the dynamic equations of a manipulator, the basic form of the manipulator equations, and the equations for the two-degree-of-freedom model used in the control algorithm comparisons discussed later in this report.

Performance requirements are the final area of concern in establishing the framework of the manipulator control problems. Typical requirements and their impact on formulation of the control problem are discussed in the final section.

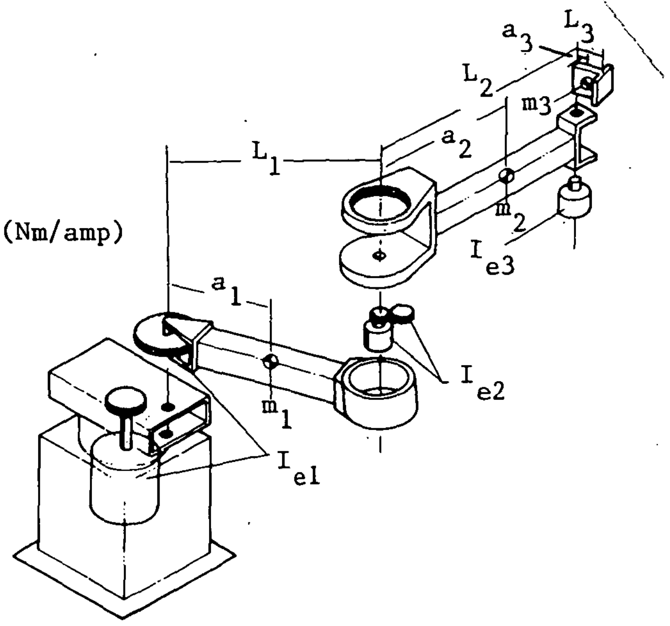
Manipulator dynamics. - The problem of manipulator control is made considerably more difficult by the nonlinear, highly coupled nature of the dynamic description of a general manipulator system.

The dynamic equations for a three-link planar arm are shown in Figure 24(a) and (b). These equations are included to demonstrate that, even for a relatively simple dynamic system, the dynamic equations become complex very quickly.

Performance requirements. - Performance requirements for a manipulator can take several forms. In the simplest case the requirement consists of tracking some reference trajectory in space with the end-effector. A time-space trajectory includes the possibility of position, velocity, and acceleration constraints. Because these constraints are usually specified in terms of a Cartesian work frame, and the control is usually in terms of measured joint variables, coordinate transformations are required to relate the two.

A second general class of requirements can be stated in terms of forces or torques applied at the end-effector. As in the case of position and attitude relating to joint positions, end-effector forces and torques are related by coordinate transformations to joint torques or forces [Wu 1982].

To perform a general task (peg-in-the-hole insertion for example), both position and force constraints must be met simultaneously. This subject is discussed in the final section in greater detail.



Definition of Variables

Motor Parameters

η_i	$\triangleq$	$\dfrac{\text{Motor Shaft Angle}}{\text{Joint Angle}}$
K_{Ti}	$\triangleq$	Torque Constant Of The Motor (Nm/amp)
K_i	$\triangleq$	Amplifier Gain ($V/_V$)
V_i	$\triangleq$	Amplifier Input Voltage (V)
K_{Bi}	$\triangleq$	Back emf Constant (V/rad/s)
R_{ai}	$\triangleq$	Armature Resistance (ohms)

LINK PARAMETERS

m_i	$\triangleq$	Mass Of Link (kg)
a_i	$\triangleq$	Distance from Joint To Center Of Gravity (cg) Of Link (m)
I_i	$\triangleq$	Moment Of Inertia Of Link about cg (kg-m^2)
L_i	$\triangleq$	Length Of Link (m)
I_{ei}	$\triangleq$	Effective Moment Of Inertia Of The Gears and The Rotating Portions Of The Motor, Measured At The Joint (kg-m^2)

Figure 24(a). - Three-link planar arm.

"Page missing from available version"

pgs 67 and 68

Although simplistic, this system retains the basic difficulties of a higher degree-of-freedom system as evidenced by the nonlinear terms. The only term missing from this equation that is found in more general systems is a force term attributable to Coriolis acceleration. The Coriolis term is manifested as the product of angular velocities, $(\dot{\theta}_i, \dot{\theta}_j)$ and is therefore very similar to the centrifugal terms; hence, no generality is lost in a controller designed for the planar system. The basic form of the controllers is shown in Figure 26.

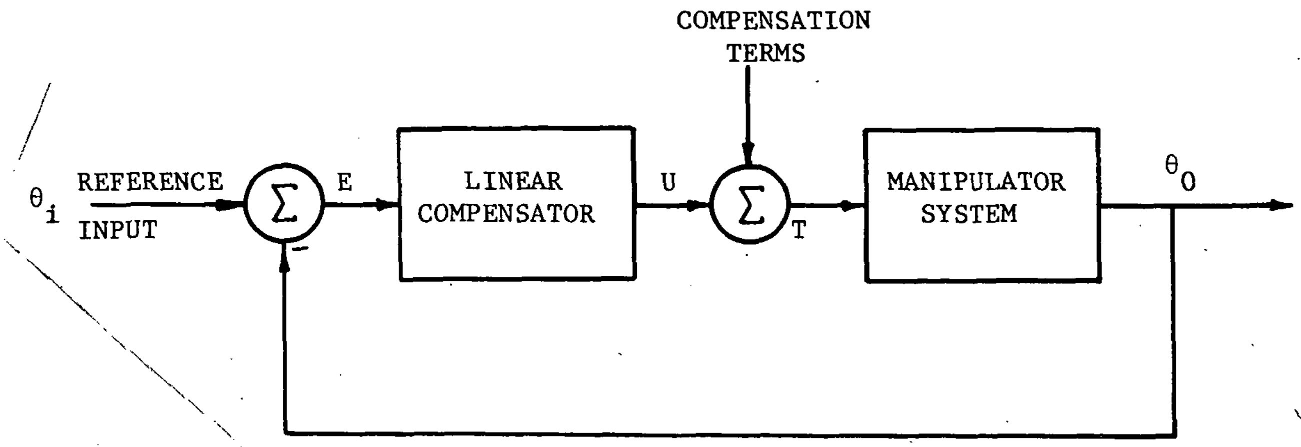


Figure 26. - General controller structure.

The compensation terms are discrete approximations to the nonlinear terms in the manipulator dynamics. Typical approximations used for compensation are shown in Table III.

Table III - NONLINEAR APPROXIMATIONS

	ACTUAL	DISCRETE APPROXIMATION
T_A	$m_2a_2L\cos(\theta_2-\theta_1)\ddot{\theta}_2$	$m_2a_2L\cos(\theta_2(k)-\theta_1(k))\dfrac{(\theta_2(k)-2\theta_2(k-1)+\theta_2(k-2))}{T^2}$
T_G	$(m_1a_1+m_2L)g\sin\theta_1$	$(m_1a_1+m_2L)g\sin\theta_1(k)$
T_C	$m_2a_2L\sin(\theta_1-\theta_2)\dot{\theta}_2^{\,2}$	$m_2a_2L\sin(\theta_1(k)-\theta_2(k))\dfrac{(\theta_2(k)-\theta_2(k-1))^2}{T}$

These nonlinear compensation terms are meant to provide "correcting" torques. In the ideal case these corrections are exact and the dynamic equations can be rewritten as:

$$(J)\ddot{\underline{\theta}} + (K)\dot{\underline{\theta}} + \underline{f}_g(\theta_1,\theta_2) + \underline{f}_c(\theta_2,\theta_1,\theta_i^2) = \underline{T}_E + \underline{T}_G + \underline{T}_A,$$

where

$$J_{12}\ddot{\theta}_2 = T_{A1} \qquad J_{21}\ddot{\theta}_1 = T_{A2}$$

$$\underline{f}_{g1} = T_{G1} \qquad f_{g2} = T_{G2}$$

Canceling these equalities leads to two decoupled linear systems,

$$J_{11}\ddot{\theta}_1 + K_1\dot{\theta}_1 = T_{E1}$$

$$J_{22}\ddot{\theta}_2 + K_2\dot{\theta}_2 = T_{E2}$$

Any conventional linear, discrete design technique may then be used to design the linear compensator shown in Figure 30. Basically, these techniques are:

1) Pole-zero compensator in conjunction with root locus using pole placement:

2) Pole-zero compensator corresponding to lead, lag, and lead-lag designs in the frequency domain using gain margin, phase margin, resonance peak, and bandwidth specifications:

3) PID design leading to closed loop pole placement.

Items 1) and 2) are shown in Figure 27; Item 3) is shown in Figure 28.

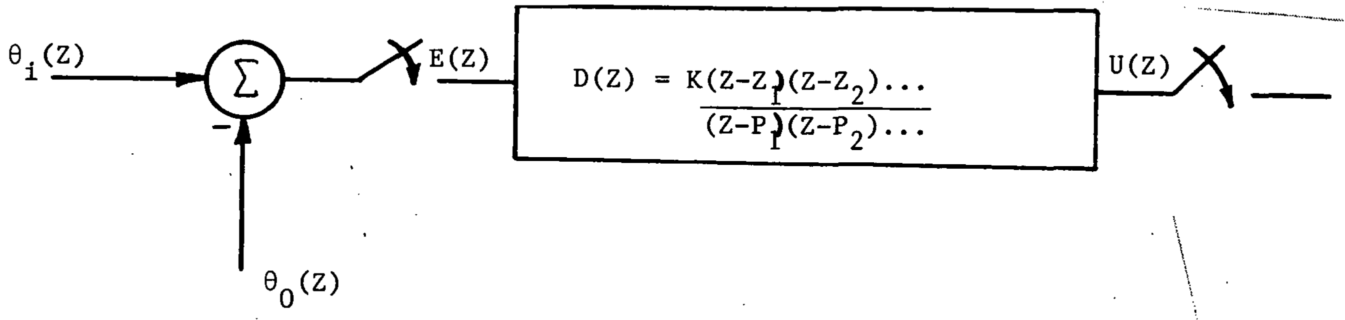


Figure 27. - Pole-zero compensator.

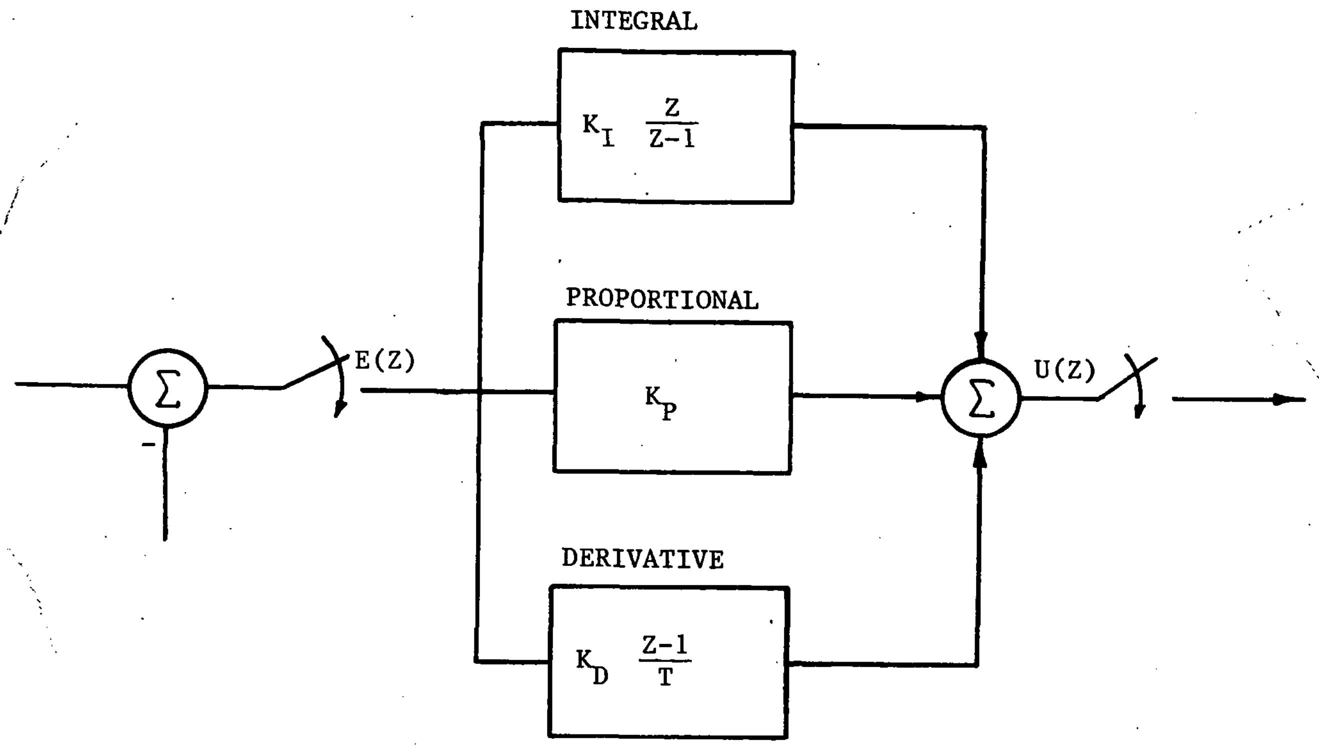


Figure 28. - PID controller.

Design techniques for selected compensators are described in more detail in the following paragraphs. Two key questions must be examined:

1) What impact does neglecting compensation terms have on control system performance?

2) What impact do additional loads with nonadjustable compensation have on performance?

As shown previously, the use of nonlinear compensation terms simplifies the problem to a standard, linear, digital design. Therefore, the first step in the design process is to develop a discrete model for the decoupled plant. Taking the Laplace transform of each term

$$(J_{11}S^2 + K_1S)\theta_{01}(S) = T_{E1}(S)$$

$$(J_{22}S^2 + K_2S)\theta_{02}(S) = T_{E2}(S)_1$$

and assuming the use of a zero-order hold device

$$H_{ZOH}(S) = \frac{1 - e^{-TS}}{S}$$

leads to the required z-transform

$$G(Z) = \frac{Z-1}{Z} \; Z\left(\frac{1}{S^2(JS+K)}\right)$$

The required Z-transform [Franklin 1981]

$$Z\left(\frac{a}{s^2(s+a)}\right) = \frac{Z((aT - 1 + e^{-aT})Z + (1 - e^{-aT} - aTe^{-aT}))}{a(Z-1)^2(Z - e^{-aT})} .$$

yields

$$G(Z) = \frac{1}{K}\,\frac{Z-1}{Z}\,Z\left(\frac{K/J}{S^2(S+K/J)}\right)$$

$$= \frac{J}{K^2}\;\frac{(aT - 1 + e^{-aT})Z + (1 - e^{-aT} - aTe^{-aT})}{(Z-1)(Z - e^{-aT})} \qquad a = K/J$$

This equation is further modified so that the impulse response of the discrete model matches that of the continuous system. In other words,

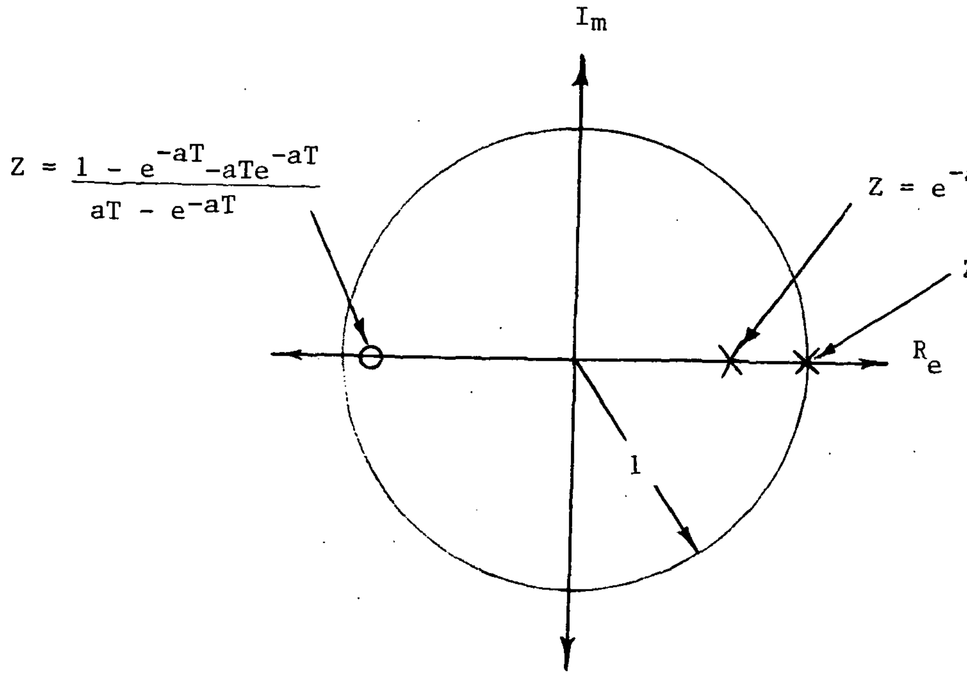
$$\lim_{S\to 0} S\,G(S) = \lim_{Z\to 1}(Z-1)G(Z),$$

by the final value theorems for both Laplace and Z transforms. This requires an additional factor of 1/T. Therefore,

$$G(Z) = \frac{1}{aKT}\;\frac{(aT - 1 + e^{-aT})Z + (1 - e^{-aT} - aTe^{-aT})}{(Z-1)(Z - e^{-aT})} .$$

$$= \frac{(aT - 1 + e^{-aT})}{aKT}\;\frac{Z + ((1 - e^{-aT} - aTe^{-aT})/(aT - 1 + e^{-aT}))}{(Z-1)(Z - e^{-aT})}$$

This yields a pole-zero diagram in the Z-plane as shown in Figure 29.



72

Figure 29. - System pole-zero diagram.

There are two basic approaches for the design of the compensator shown in Figure 29: root locus or frequency domain design in the W-plane [Franklin 1981].

In the root-locus approach the compensator is used to alter the loop transfer function through pole cancellation, followed by a gain adjustment to achieve the desired closed-loop pole placement. This is shown in Figure 30.

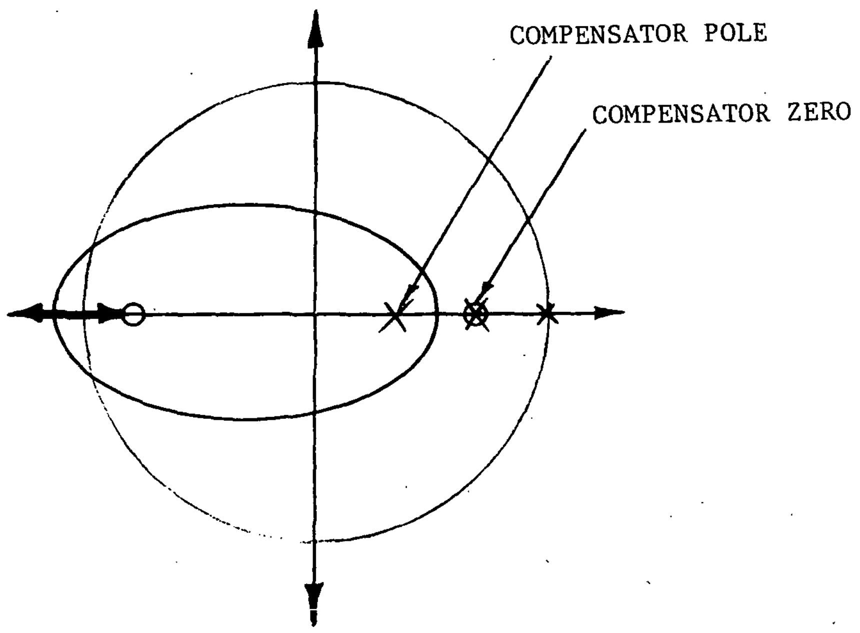


Figure 30. - Root locus with pole cancellation.

The second compensator design technique explored uses the PID controller shown in Figure 28. The three terms in this controller can be combined to form a general G(Z)

$$G(Z) = \frac{K_I Z + K_P(Z - 1) + (K_D/T)(Z - 1)^2}{(Z - 1)}$$

The three adjustable parameters, K_I, K_P, and K_D allow arbitrary placement of the poles of the closed-loop system.

The PID controller has another important property with regard to disturbance rejection (Fig. 31).

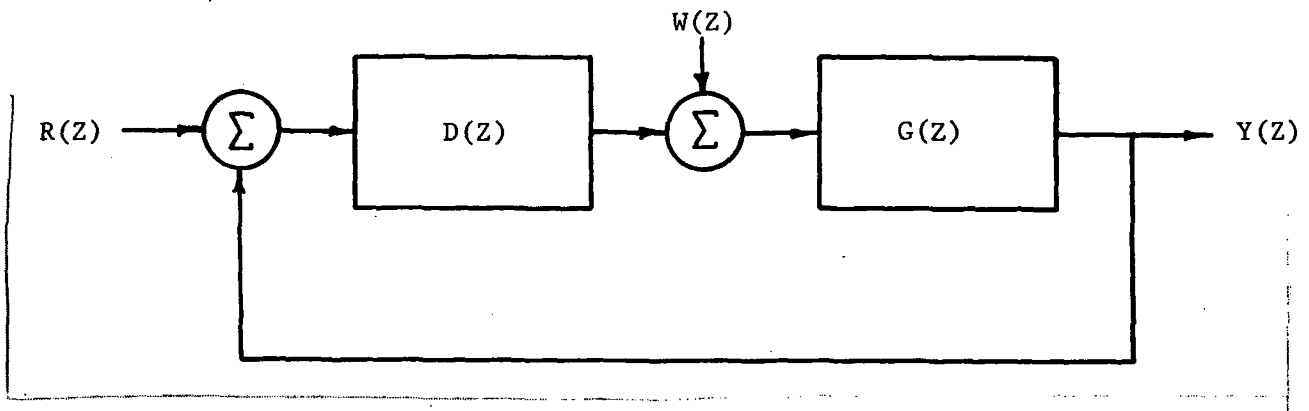


Figure 31. - General control loop with disturbance, W(Z).

The input/output transfer function is

$$\frac{Y(Z)}{R(Z)} = \frac{D(Z)G(Z)}{1+D(Z)G(Z)}$$

The disturbance transfer function is

$$\frac{Y(Z)}{W(Z)} = \frac{G(Z)}{1+D(Z)G(Z)}$$

Let, $G(Z) = \frac{n_1(Z)}{d_1(Z)}$; $D(Z) = \frac{n_2(Z)}{Z - 1}$, then , then

$$Y(Z) = W(Z) \frac{\frac{n_1(Z)}{d_1(Z)}}{1 + \frac{n_1(Z)n_2(Z)}{(Z - 1)d_1(Z)}}$$

$$= W(Z) \frac{(Z - 1)(n_1(Z))}{(Z - 1)d_1(Z) + n_1(Z)n_2(Z)}$$

This shows that if W(Z) is a step disturbance

$$W(Z) = \frac{Z}{Z-1}$$

then

$$Y(Z) = \frac{Z(n_1(Z))}{(Z - 1)d_1(Z) + n_1(Z)n_2(Z)}$$

If the denominator has roots inside the unit circle, the steady-state response to a step disturbance is zero; i.e., there is no steady-state "offset." This is attributable to the pure integrator in the controller. In contrast, assume (Z-1) is replaced by some general $d_2(z)$ in the disturbance transfer function. Then for a step disturbance

$$Y(Z) = \frac{Z}{Z-1} \; \frac{d_2(Z)n_1(Z)}{d_1(Z)d_2(Z) + n_1(Z)n_2(Z)}$$

The steady-state contains a step component

$$Y_{step}(Z) = \lim_{Z \to 1} \frac{Zd_2(Z)n_1(Z)}{d_1(Z)d_2(Z) + n_1(Z)n_2(Z)} \neq 0$$

This implies that there will be a steady-state error to a step disturbance. The physical interpretation of these results is straightforward. At equilibrium, in the non-PID case, a non-zero error is required to provide the torque necessary to maintain equilibrium. In the PID case, the error can be zero while the output from the integrator provides the torque to counter the disturbance. From the standpoint of a manipulator system this is an interesting property because gravity disturbances are essentially step disturbances around a given operating region.

Resolved acceleration control (RAC). - The RAC is a method of position control of a manipulator. The algorithm uses the position and orientation, the velocity, and the acceleration of the hand (or end-effector). The desired position and orientation, velocity, and acceleration of the hand are specified at each time step, and the control algorithm determines the joint positions, velocities, and accelerations required to satisfy the specified conditions. Finally, the input forces and torques to be applied to the joints are calculated using the "inverse problem" technique [Luh 1980a].

One advantage of the RAC method is that the forces and torques are not obtained by the dynamics equations; rather they are calculated as a function of the desired and actual joint positions, velocities, and accelerations, using the Newton-Euler formulation of the manipulator equations. The method successively transforms the velocities and accelerations from the base to the hand, using moving coordinate system relationships. The input forces and torques are computed for each link (working back from the hand to the base). This scheme allows much faster computation than the conventional technique based on the Lagragian formulation of the dynamics [Luh 1980b].

Resolved motion force control (RMFC). - RMFC is a position control method developed for robot manipulators [Wu 1982]. Rather than controlling the joint positions and torques (as do classical control techniques), RMFC controls Cartesian positions and forces. This method has the advantage of automatically compensating for the changing configuration of the manipulator, gravity forces, and the internal friction of the manipulator. Another advantage of RMFC is that the control method can be extended to more than six degrees of freedom without increasing the computational complexity.

To effect control of a position manipulator, it is usually desired to drive it to follow a given trajectory. Paul's method of defining a manipulator's position and orientation, the relationship between Cartesian forces and manipulator joint torques, the specification of a manipulator's position trajectory, and the RMFC algorithm are discussed below.

The position and orientation of an n-link manipulator can be described by a 4x4 transformation matrix

$$T_n = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where $P = (p_x\ p_y\ p_z)^T$ is the position and n_x, n_y, n_z, o_x, o_y, o_z, a_x, a_y and a_z are the direction cosines defining the manipulator's orientation.

The matrix T_n is the product of all n transformation matrices, A_i, i = 1 to n, describing the position and orientation of link i relative to link i-1

$$T_n = A_1\ A_2\ \ldots\ A_n$$

The linear and angular velocities of the terminal link are a function of the Jacobian and the joint rates

$$\underline{\dot{x}} = [J]\underline{\dot{q}}$$

where $\underline{\dot{x}} = \begin{bmatrix} v \\ w \end{bmatrix}$ $\begin{matrix} = \text{linear velocity} \\ = \text{angular velocity} \end{matrix}$

The terminal link static forces $F_n = [F_x\ F_y\ F_z\ M_x\ M_y\ M_z]^T$ and the static joint torques $F_t = [f_1\ f_2\ \ldots\ f_{n-1}\ f_n]^T$ are related

$$\underline{F_t} = [J]^T\underline{F_n}$$

where

$[J]^T$ is the transpose of the Jacobian; the differential change in position and orientation of T_n as a function of all n joint angles

$f_1, \ldots f_n$ are the joint torques; F_x, F_y, and F_z are the Cartesian forces at the end-effector

M_x, M_y, and M_z are the moments at the end-effector

A time-based position trajectory can be defined as:

$$T_n(t+\ \ t) = \begin{bmatrix} 1 & -w_z(t) & w_y(t) & v_x(t) \\ w_z(t) & 1 & -w_x(t) & v_y(t) \\ -w_y(t) & w_x(t) & 1 & v_z(t) \\ 0 & 0 & 0 & 1 \end{bmatrix} \Delta t$$

where v(t) is the linear velocity and w(t) is the angular velocity $T_n(t)$, v(t), and w(t) must be continuous.

The basic idea of RMFC is to calculate the joint torques required to control the Cartesian motion of the manipulator. The method is based on the relationship between the joint torques and the terminal link forces. The control is divided into two parts. The first part deals with position control of the terminal link. The second part of resolved motion force control calculates the joint torques required to obtain the desired terminal link forces and moments that were determined in the first part.

For the first part of the control it is assumed that all links except the terminal link and the load are massless. Also, the center of mass of the load is located at the origin of the terminal link coordinate system, and the coordinate axes of the load are aligned with the terminal link axes. Because the manipulator's links are not massless, the second part of the RMFC scheme compensates for these errors using a method called force convergent control. The technique uses the Robbins Monro stochastic approximation method to determine the joint torques required to produce the desired Cartesian forces and moments of the center of mass of the load.

The desired position and orientation can be expressed in the form of a transformation matrix T_{nd}. The actual position and orientation T_{na} is determined according to equation for T_n above. The Cartesian position and orientation error

$$X_e = X_d - X_a = [d_x \; d_y \; d_z \; \delta_x \; \delta_y \; \delta_z]^T$$

can be written as

$$T_{nd} = T_{na} \begin{bmatrix} 1 & -\delta_z & \delta_y & d_x \\ \delta_z & 1 & -\delta_x & d_y \\ -\delta_y & \delta_x & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The differential transformation matrix is obtained as follows. The transformation matrices Tx, Ty, and Tz defining a rotation about the x, y and z axes, respectively are given by

$$T_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_y = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_z = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For a differential change $d\theta$;

$$\lim_{\theta \to 0} \sin\theta = d\theta$$

$$\lim_{\theta \to 0} \cos\theta = 1$$

Substituting these expressions into T_x, T_y, and T_z, multiplying the resulting matrices together, and neglecting second order terms gives

$$T = T_{\delta x} T_{\delta y} T_{\delta z} = \begin{bmatrix} 1 & -\delta_z & \delta_y & 0 \\ \delta_z & 1 & -\delta_x & 0 \\ -\delta_y & \delta_x & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Multiplying T (on the left or right) by a matrix T_d defining a differential translation along the x, y, and z axes gives

$$T = \begin{bmatrix} 1 & -\delta_z & \delta_x & d_x \\ \delta_z & 1 & -\delta_x & d_y \\ -\delta_y & \delta_x & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This is precisely the differential transformation T_{nd}. It should be noted that this matrix is independent of the order of rotations.

Explicit expressions for d_x, d_y, d_z, δ_x, δ_y, and δ_z of T_{nd} are obtained as follows.

Define T_{na} and T_{nd} as

$$T_{na} = \begin{bmatrix} n_{ax} & o_{ax} & a_{ax} & p_{ax} \\ n_{ay} & o_{ay} & a_{ay} & p_{ay} \\ n_{az} & o_{az} & a_{az} & p_{az} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{nd} = \begin{bmatrix} n_{dx} & o_{dx} & a_{dx} & p_{dx} \\ n_{dy} & o_{dy} & a_{dy} & p_{dy} \\ n_{dz} & o_{dz} & a_{dz} & p_{dz} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rewriting gives

$$T_\delta = \begin{bmatrix} 1 & -\delta_z & \delta_y & d_x \\ \delta_z & 1 & \delta_x & d_y \\ -\delta_y & \delta_x & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = T_{na}^{-1} T_{nd}$$

where

$$T_{na}^{-1} = \begin{bmatrix} n_{ax} & n_{ay} & n_{az} & -p_a \cdot n_a \\ o_{ax} & o_{ay} & o_{az} & -p_a \cdot o_a \\ a_{ax} & a_{ay} & a_{az} & -p_a \cdot a_a \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Premultiplying T_{nd} by T_{na}^{-1} and equating elements of the resulting matrix with the elements of T gives

$$X_e = \begin{bmatrix} d_x \\ d_y \\ d_z \\ \delta_x \\ \delta_y \\ \delta_x \end{bmatrix} = \begin{bmatrix} n_a \cdot (p_d - p_a) \\ o_a \cdot (p_d - p_a) \\ a_a \cdot (p_d - p_a) \\ a_a \cdot o_d \\ n_a \cdot a_d \\ o_a \cdot n_d \end{bmatrix}$$

where n_d, o_d, a_d, p_d, and n_a, o_a, a_a, p_a are the column vectors of T_{nd} and T_{na}, respectively.

The actual Cartesian velocity

$$\dot{\underline{x}}_a = [v_{xa}\ v_{ys}\ v_{za}\ w_{xa}\ w_{ya}\ w_{za}]^T \text{ at } T_n$$

is given by

$$\dot{\underline{x}}_a = [J]\dot{\underline{q}}_a$$

where [J] is the Jacobian of the manipulator and $\dot{q}_a$ are the actual joint velocities.

Rewriting the equation for $T_n(t+\Delta t)$ gives an expression for the desired Cartesian velocity $\dot{\underline{X}}_d = [V_{xd}\ \ v_{yd}\ \ v_{zd}\ \ w_{xd}\ \ w_{yd}\ \ w_{zd}]^T$:

$$\begin{bmatrix} 1 & -w_{zd} & w_{yd} & v_{xd} \\ w_{zd} & 1 & -w_{xd} & v_{yd} \\ -w_{yd} & w_{xd} & 1 & v_{zd} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \frac{1}{\Delta t}\,[T_{nd}^{-1}(t) * T_{nd}(t + \Delta t)]$$

where Δt is the sample time

Performing the multiplication on the right-hand side above and equating elements of the resulting matrix to the elements of the matrix on the left side yields

$$\dot{\underline{X}}_d(t) = \begin{bmatrix} v_{xd}(t) \\ v_{yd}(t) \\ v_{zd}(t) \\ w_{zd}(t) \\ w_{yd}(t) \\ w_{zd}(t) \end{bmatrix} = \begin{bmatrix} [n_d(t) \cdot [p_d(t + \Delta t) - p_d(t)]]/\Delta t \\ [o_d(t) \cdot [p_d(t + \Delta t) - p_d(t)]]/\Delta t \\ [a_d(t) \cdot [p_d(t + \Delta t) - p_d(t)]]/\Delta t \\ [a_d(t) \cdot o_d(t + \Delta t)]/\Delta t \\ [n_d(t) \cdot a_d(t + \Delta t)]/\Delta t \\ [o_d(t) \cdot n^d(t + \Delta t)]/\Delta t \end{bmatrix}$$

The Cartesian velocity error can be calculated by subtracting

$$\underline{\dot{X}}_e = \underline{\dot{\bar{X}}}_d - \underline{\dot{\bar{X}}}_a$$

The desired Cartesian acceleration can then be estimated

$$\underline{\ddot{X}}_d(t) = (\dot{X}_d(t + \Delta t) - \underline{\dot{X}}_d(t))/\Delta t$$

The position control loop for the system uses position and derivative feedback in addition to the desired acceleration. The actual acceleration is given by the expression

$$\ddot{x}_a(t) = K_v\dot{x}_e(t) + K_p x_e(t) + \ddot{x}_d(t)$$

where:

K_p is the position gain K_v is the velocity gain.

Differentiating the equation for Cartesian error and substituting the result into the above gives

$$\ddot{X}_e(t) + K_v\dot{X}_e(t) + K_p X_e(t) = 0$$

To drive X_a to converge to X_d, K_v and K_p must be chosen so the roots of this equation have negative real parts.

The desired Cartesian forces and moments can be calculated as $\underline{F}_d(t) = M\underline{\ddot{X}}_a(t)$

where

$$M = \begin{bmatrix} m_t & 0 & 0 & 0 & 0 & 0 \\ 0 & m_t & 0 & 0 & 0 & 0 \\ 0 & 0 & m_t & 0 & 0 & 0 \\ 0 & 0 & 0 & I_x & 0 & 0 \\ 0 & 0 & 0 & 0 & I_y & 0 \\ 0 & 0 & 0 & 0 & 0 & I_z \end{bmatrix}$$

m_t is the total mass of the load

I_x, I_y, I_z are the second moments of inertia about the coordinate axes of the load.

The applied joint torques required to produce the desired Cartesian forces and torques can be calculated

$$\underline{F}_t = [J]^T\underline{F}_d$$

When the mass of the terminal link and the load approaches the mass of the remaining portion of the manipulator, the control algorithm will not converge to the desired Cartesian forces and torques. Force convergent control compensates for this error.

Force convergent control uses Robbins-Monro stochastic approximation to calculate the applied Cartesian force $\underline{F}_a$ (and the corresponding joint torques) required to drive the observed Cartesian force $\underline{F}_o$ to converge to the desired Cartesian force $\underline{F}_d$.

The Robbins-Monro method, which finds the root of a regression function, can be summarized as follows. Let be the parameter vector to be estimated and the $\underline{Z}_k$ be the observation at time K. Then the new estimate at time K+1 is given by

$$\underline{\theta}_{K+1} = \underline{\theta}_K - b_K \underline{Z}_K$$

The coefficient $\underline{b}_k$ is a sequence of positive numbers with the following properties

$$\lim_{K\to\infty} b_K = 0$$

$$\sum_{K=1}^{\infty} b_K = \infty$$

$$\sum_{K=1}^{\infty} b_K^2 < \infty$$

The first equation ensures that the estimate will settle down in the limit. The second ensures that the estimate does not settle down before reaching the root. The third ensures that the variance of the accumulated noise is finite so the effect of the noise can be corrected [Fukunaga 1972].

The manipulator is modeled as an unknown function. The input of the function at each sample time is $\underline{F}_a$, the applied force. The function's output is the observed force $\underline{F}_o$ at the load. This can be stated as

$$\underline{F}_0 = F(\underline{F}_a)$$

The method can be described as three basic steps. First, define an error tolerance for the force

$$d\underline{F} = [dF_1\ dF_2\ dF_3\ dF_4\ dF_5\ dF_6]^T$$

This is the acceptable error between the observed Cartesian force $\underline{F}_o$ and the desired Cartesian force $\underline{F}_d$.

Next, initialize the applied force $\underline{F}_a$ at t = 0 to the Cartesian force caused by the gravity force of the load. Finally, at each sample time, perform the following operations:

1) Calculate $\underline{F}_d$ using $\underline{F}_d(t) = M\ddot{\underline{X}}_a(t)$

2) Set K=0 and $\underline{F}_a(0)$ = final $\underline{F}_a$ of previous sample time;

3) Apply the joint torques $\underline{F}_t = [J]^T \underline{F}_a(K)$;

4) Measure the observed force $\underline{F}_o(K)$;

5) Calculate the Cartesian force error $\delta F(K) = \underline{F}_d - \underline{F}_o(K)$

6) Calculate

   a) If $|\delta F_i| > dF_i$, $i = 1, 2, \ldots, 6$, compensate $\underline{F}_a$ according to the following equation

   $\underline{F}_a(K+1) = \underline{F}_a(k) + b_k \, \delta F(k)$

   where $b_K = 1/(K+1)$ for $K = 0, 1, 2, \ldots$

   b) Increment K by 1,

   c) Return to step 3 (apply new joint torques);

7) If $|\delta F_i| < dF_i$, $i = 1, 2, \ldots, 6$, $\underline{F}_0(K)$ has converged to the desired force $\underline{F}_d$. Therefore, apply the joint torques in step 3) for the remaining portion of the sample time.

It should be noted that in the simulations documented in [Wu 1982], the above algorithm was modified to perform a finite number of iterations, N, rather than using the force error criterion to test convergence. As N is increased, the accuracy of the algorithm increases. It should be noted further that N cannot exceed $\Delta t/h$, where Δt is the sample time (in seconds) and h is the time (in seconds) required to perform the operations in FCC.

The algorithms discussed so far for manipulator control all rely on the assumption that the manipulator dynamic parameters are known for use in control law formulation. Although the general form of the manipulator equations is well known, the precise coefficients (specifically the inertia matrix) will generally be time-varying. One technique to overcome the effects of the time-varying manipulator dynamics on system performance is to make the controller adaptive. While the term adaptive has been used loosely in many applications, it will be defined in this discussion to mean "parameter adaptive" control.

A parameter adaptive controller adjusts the coefficients of a variable "filter." Figure 32 is a block diagram of an adaptive controller.

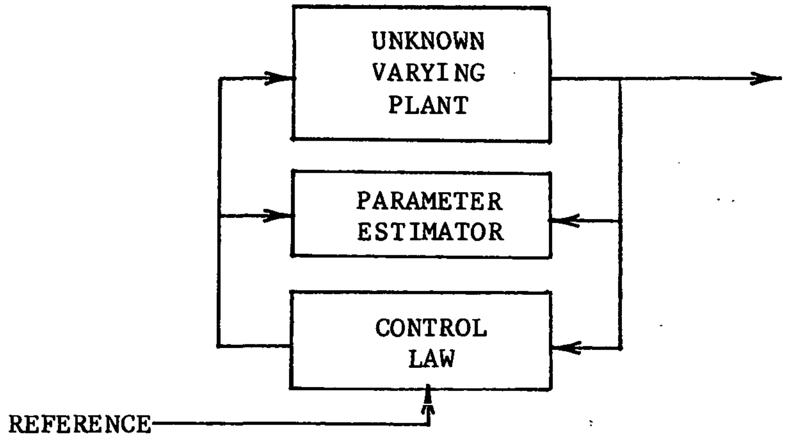


Figure 32. - General adaptive controller.

As shown, it is assumed that the plant to be controlled is either initially unknown or time-varying. The parameter adaptive controller consists of two primary blocks--the parameter estimation block and the control law block.

Parameter estimation. - The parameter estimation block determines, using sensor data, a set of coefficients that describes the system to be controlled. This definition is purposely vague because a large number of parameter estimation techniques exist. A typical parameter estimation block is shown in Figure 33.

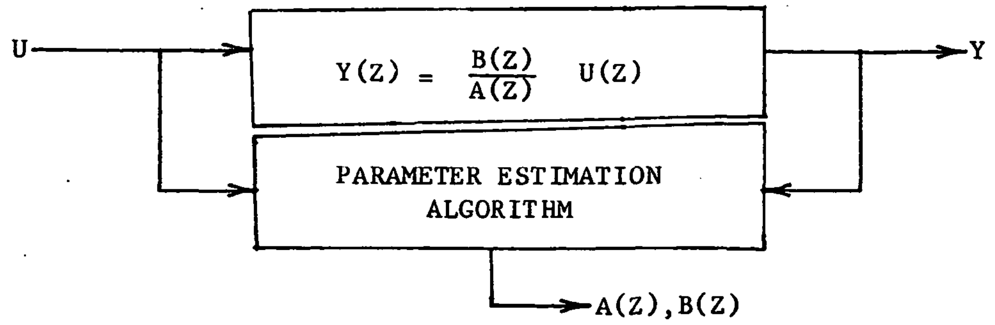

U
$Y(Z) = \frac{B(Z)}{A(Z)} U(Z)$
Y
PARAMETER ESTIMATION ALGORITHM
A(Z),B(Z)

Figure 33. - General parameter estimation.

If the plant is a linear system, it can be expressed as a discrete model as

$$Y(K+1) = a_1 Y(K) + \dots + a_n Y(K-n+1) + b_0 U(K) + \dots + b_m U(K-m)$$

The parameter estimation algorithm operates on past values of input and output data to generate an estimate of the unknown parameters - $a_1, \dots a_n$, and $b_0, \dots b_m$.

Techniques for parameter estimation have been widely studied both for control and signal-processing applications. Algorithms are available for deterministic and stochastic systems.

The parameter estimation algorithm to be discussed here is recursive least squares (RLS). It is one of the most robust estimation schemes and converges more rapidly than some of the simpler methods. Recursive least squares is classified as an "equation error" parameter estimation scheme because of the basic structure of the algorithm. Most parameter estimation techniques are of the form [Goodwin 1982]

$$\underline{\hat{\theta}}(K+1) = \underline{\hat{\theta}}(K) + \Gamma e(K)$$

In "equation error" estimation schemes, the error is defined as

$$e(K) = Y(K) - \underline{\phi}^T \underline{\theta}(K)$$

where

$$\underline{\phi} = [Y(K-1), \dots, Y(K-n+1), U(K-1), \dots, U(K-m-1)]^T$$
$$\underline{\hat{\theta}} = [a_1, \dots a_n, b_0, \dots b_m]^T$$

For the true parameters, $e(K) = 0$. Thus the name "equation error." $e(K)$ is a measure of how well the parameter estimates satisfy the governing equation. If $e(K) \neq 0$, the parameter estimator will continue to update the estimate. The "Γ" term is what differentiates parameter estimation schemes. It can be thought of as a "gain" term for the estimation process. For the recursive least squares estimator, is chosen to be [Goodwin 1982] and [Franklin 1981]

$$\Gamma = \frac{a(k)\ P(K-d-1)\ \phi(K-d)}{1 + a(K)\ \phi(K-d)^T P(K-d-1)\ \phi(K-d)}$$

and

$$P(K-d) = P(K-d-1) - \frac{a(K)\ P(K-d-1)\ \phi(K-d)\ \phi(K-d)^T P(K-d-1)}{1 + a(K)\ \phi(K-d)^T P(K-d-1)\ \phi(K-d)}$$

To start the RLS algorithm,

$P(\ -d-1) = \ \varepsilon_0 I$

where $\varepsilon >> 1$ and

I is the identity matrix

$0 < a(K) < 2$
$\hat{\underline{\theta}}(0) = [0,\ldots,0,1,0,\ldots 0]$

where the "1" corresponds to the parameter b_0.

If a good "first guess" of the parameters is available, it can be used for $\hat{\underline{\theta}}(0)$. The estimate of the coefficient b_0 should not be initialized to zero because this results in division of 0.

The coefficient $a(K)$ is usually chosen to be unity; however, in cases where $\phi(K-d)^T\ P(K-d-1)\ \phi(K-d)$ (in the denominator of Γ) is close to minus 1, $a(K)$ should be chosen to avoid division by a number close to zero.

Once an estimate of the parameters has been obtained, it is used in formulation of the control law. The most general control form is model-reference control. In this case the controller functions to make the closed-loop system match a desired transfer function (or matrix). A special case of model-reference control is "one-step ahead optimal" or minimum-variance control. These control strategies are discussed in more detail in the next section.

Linear Adaptive Control. - A linear, deterministic system is described by an auto-regressive moving average (ARMA) equation of the form

$$Y(K+1) = a_1 Y(K) + \ldots + a_n Y(K-n+1) + b_0\ U(K) + \ldots + b_m U(K-m)$$

To formulate a "one step-ahead" or "deadbeat" controller, it is assumed that the reference signal is known one step in advance. That is, Y*(K+1) must be known at the time K. The control law for the parameter adaptive controller is obtained by setting Y(K+1) = Y*(K+1) and solving for U(K) [Goodwin 1983].

This yields

$$U(K) = \frac{1}{b_0}[\ -a_1\ Y(K) - \ldots - a_n Y(K-n+1) - b_1 U(K-1) - \ldots - b_m\ U(K-m) + Y^*(K+1)]$$

This can also be written in vector notation as

$$U(K) = \underline{\phi}(K)^T \underline{\theta}(K)$$

where $\underline{\phi}(K) = [\ -Y(K)\ \ldots\ -Y(K-n+1) - U(K-1) \ldots -U(K-m)\ Y^*(K+1)]^T$

$$\underline{\theta}(K) = \left[\frac{a_1}{b_0} \ldots \frac{a_n}{b_0}, \frac{b_1}{b_0} \ldots \frac{b_m}{b_0}, \frac{1}{b_0}\right]$$

The vector $\phi(K)$ is referred to as the regression vector and $\theta(K)$ is the parameter vector. This is referred to as the linear controller form because the control law is a linear function of past Y and U values. If the a and b coefficients are known, the control for U(K) will result in Y(K+1) = Y*(K+1). Because it is assumed, however, that the system parameters are initially unknown (or time-varying in the case of manipulator control), the a and b coefficients must be estimated. The control law then becomes

$$U(K) = \phi(K)^T\ \hat{\underline{\theta}}(K)$$

where $\hat{\underline{\theta}}(K)$ is the vector of parameter estimates.

The parameter estimation algorithm used to obtain $\hat{\underline{\theta}}(K)$ is RLS as explained in the previous section.

While this form of control is appealing from the standpoint of its simplicity and apparent tracking performance, it is impractical for implementation in an actual system. The superior performance is attained at the expense of large control efforts. For practical systems this leads to saturation of control components and hence full performance is never achieved.

A more reasonable approach to the problem is to force the closed-loop system to behave as some reference model that is chosen with the limitations of amplifiers and actuators in mind. (Strictly speaking, the one-step-ahead controller is a model reference controller. The reference model is simply a pure delay block with the number of delays governed by the number of delays in the plant ARMA equation.)

The first step in the formulation of a model reference controller is to specify the structure of the reference model

$$\begin{aligned} Y^*(k+1) &= e_1Y^*(k) + \ldots + e_nY^*(k-n+1) \\ &\quad + h_0R(k) + \ldots + h_mR(k-m) \end{aligned}$$

In this case, the "e" coefficients govern the poles of the reference model, the "h" coefficients govern the zeros, and the sequence R(k) is the reference input signal.

The first step in the formulation of the model reference control law is to isolate the current input term, U(k) as before

$$\begin{aligned} U(k) = \frac{1}{b_0} [&-a_1Y(k) - \ldots - a_nY(k-n+1) - b_1U(k-1) \\ &-b_mU(k-m) + Y^*(k+1)] \end{aligned}$$

This step was sufficient to generate the control signal for the one-step-ahead controller. Now, instead, Y*(k+1) is replaced by the terms on the right-hand side of the reference model

$$\begin{aligned} U(k) = \frac{1}{b_0} (&-a_1Y(k) - \ldots a_nY(k-n+1)-b_1U(k-1)-b_mU(k-m) \\ &+e_1Y^*(k)+ \ldots +e_nY^*(k-n+1) \\ &+h_0r(k) + \ldots + h_mR(k-m)] \end{aligned}$$

This control law could be used as is by keeping track of previous Y* values. However, assume that Y_1 and not Y^*_1 is used in generating the control. Substituting and combining terms yields

$$\begin{aligned} U(k) = \frac{1}{b_0}(&e_1-a_1)Y(k) + \ldots + (e_n-a_n)Y(k-n+1) \\ &-b_1U(k-1) - \ldots -b_mU(k-m)+h_0R(k) \\ &+ \ldots + h_mR(k-m)] \end{aligned}$$

Under these conditions, this system has the compensation structure shown in Figure 34.

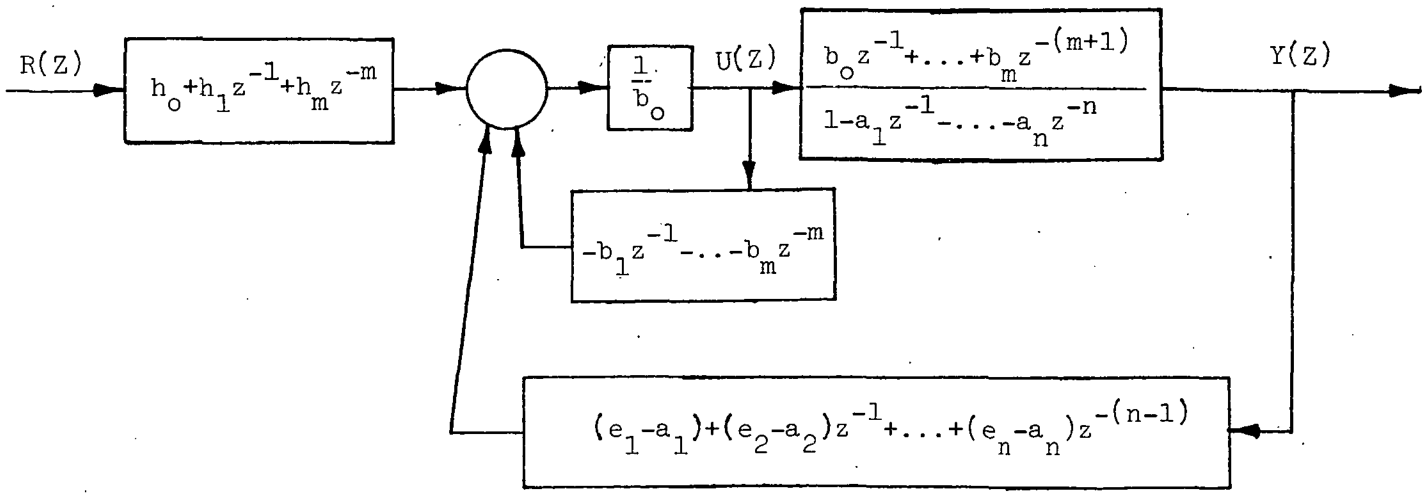

R(Z)
$h_o+h_1z^{-1}+h_mz^{-m}$
$\frac{1}{b_o}$
U(Z)
$\frac{b_oz^{-1}+\ldots+b_mz^{-(m+1)}}{1-a_1z^{-1}-\ldots-a_nz^{-n}}$
Y(Z)
$-b_1z^{-1}-\ldots-b_mz^{-m}$
$(e_1-a_1)+(e_2-a_2)z^{-1}+\ldots+(e_n-a_n)z^{-(n-1)}$

Figure 34. -Model reference control structure.

When the control law above is substituted into the original system equation, this results in:

$$U(k+1) = e_1Y(k) + .. +e_nY(k-n+1)$$
$$+h_0R(k) + .. +h_mR(k-m)]$$

Subtracting the above from the basic reference model equation and defining $E(k) = Y^*(k) - Y(k)$ yields

$$E(k+1) = e_1E(k) + ... +e_n(E(k-n+1)$$

The significance is that initial startup mismatches between Y* and Y will be regulated out as a function of the poles of the reference model. The closed-loop transfer function, utilizing this control law, is the desired reference model transfer function. (This can be shown by simple block diagram reduction of Figure 34.)

Some simulation results for a single-input, single-output linear system are shown in Figure 35. THe performance of the controller when no load change is involved is shown in Figure 35(a). Parameter convergence is rapid and tracking excellent after initial transient behavior. Figure 35(b) shows the results of a load change at 50 iterations. As expected, control degrades as reidentification takes place, but resumes one-step-ahead behavior after the parameters have settled.

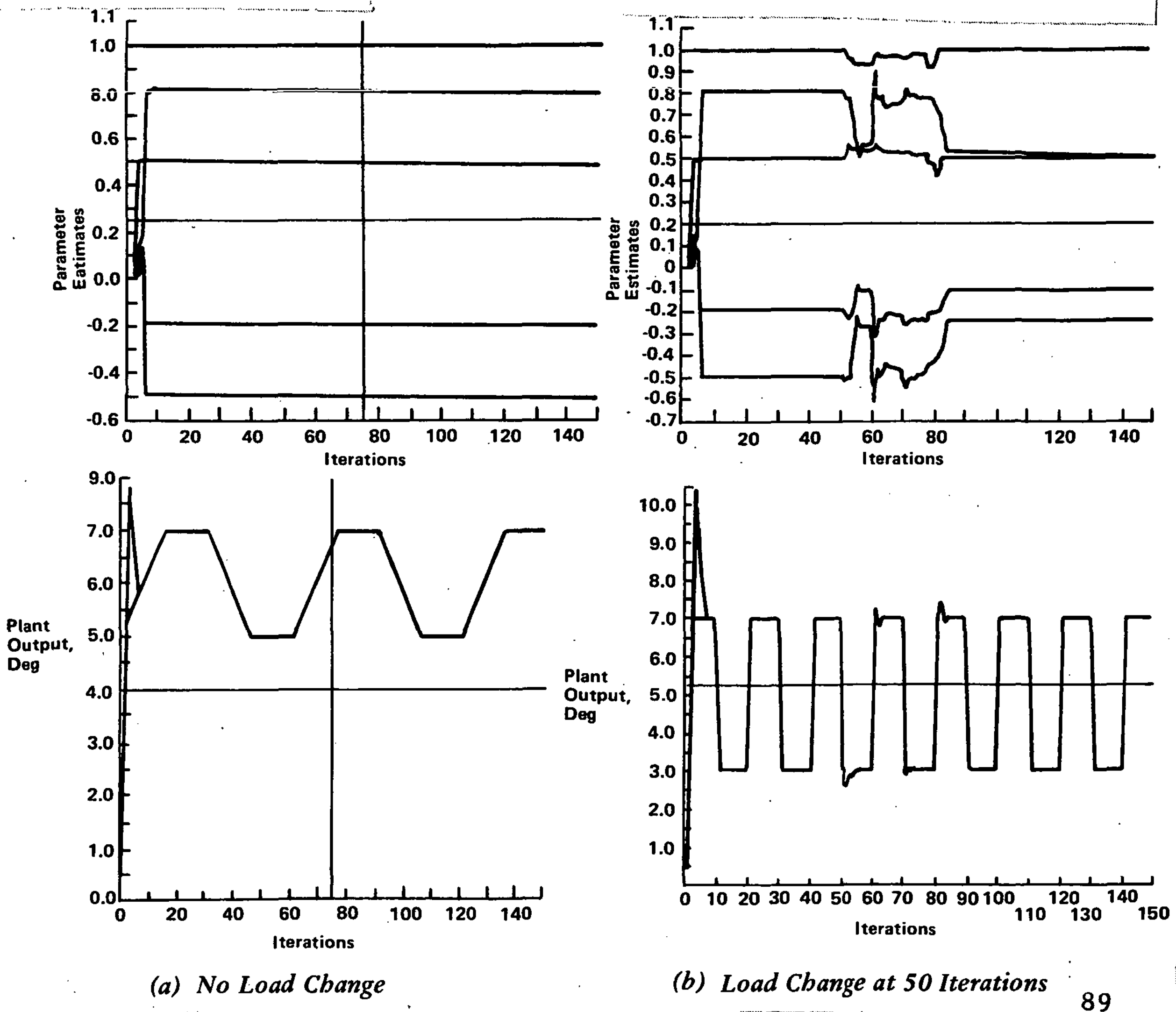


Figure 35. - Linear adaptive control simulation.

Nonlinear adaptive control. - The major body of results developed for adaptive control systems have focused on linear systems. For several years, however, researchers have addressed the problem of applying the adaptive control theory to robotic manipulators. Aside from purely theoretical interest, this pursuit is practical. With the increased interest in intelligent, flexible automation systems, manipulators are expected to be forced to perform over their entire load range, often with loads that have poorly defined mass properties. An adaptive strategy is ideally suited for this type of situation. The problem lies in extending a body of theory developed primarily for single-input single-output linear systems to the highly coupled, strongly nonlinear dynamics that characterize a manipulator.

Two of the primary approaches to adaptive control of manipulator systems will be described in greater detail. In [Leininger 1983], the problem is attacked by assuming that each joint can be adequately modeled as

$$(1 + A(Z^{-1}))Y(t) = Z^{-k}B(Z^{-1})U(t) + e(t) + d(t).$$

A and B represent polynomials in Z^{-1}, e(t) is a white noise sequence, and d(t) is a "takeup" term that accounts for coupling from other joints. A recursive least squares identification technique is used to identify the parameters of this model. The controller cancels the effect of d(t), (essentially a feedforward term), and then computes compensation terms to yield a critically damped joint response. Simulations of this approach in [Leininger 1983] demonstrate good system response in terms of output tracking, but time histories of parameter identification are not provided. Because d(t) represents the total contribution of effects from all other joints, it would seem that the identification loop would have to be very fast to adequately track this term.

[Lee 1982] proposed another approach. Rather than approximating the coupling effects at each joint, the full set of nonlinear dynamic equations are used to generate a state-space model that consists of the nonlinear equations linearized about a nominal path. An RLS identification routine is used to identify the coefficients of the linearized model. Once the state-space form is available, a variety of approaches to control exist, including pole-placement and optimal regulator theory. Although not referenced, this approach to adaptive control is reminiscent of that proposed in [Kreisselmeier 1981] for linear systems.

The nonlinear adaptive control approach that has served as the focus of this study was developed by Dr. Howard Elliott at the University of Massachusetts, Amherst. The control approach can be summarized as feedforward cancellation of nonlinearities at each joint, with subsequent model reference control of the remaining linear terms. This approach to nonlinear control is not new and apparently was first used in [Hemami 1982]. The significant contributions made by Dr. Elliott are:

1) Recursive least squares can be applied to the nonlinear manipulator dynamics without first linearizing the system of equations,

2) By assuming knowledge of primitive dynamic parameters, i.e., link lengths and the gravity coefficient g, there is a large amount of redundancy in the manipulator equations and hence the parameter estimation problem can be drastically reduced. In the case of a three-link system, Elliott's back-substitution technique reduces the parameter estimation problem from 23 parameters to three parameters per joint, for a total of nine. Other studies have shown that the estimation of three parameters can be accomplished at 100 Hz with an Intel 8086.

The use of recursive least squares to directly identify nonlinear terms was suggested in [Goodwin 1982]. As an example of this technique, consider the nonlinear plant model shown below, where $g(Y(k))$ is a nonlinear function of $Y(k)$, and the structure of the nonlinear function is known, e.g., $g(Y(k)) = \cos(Y(k))$

$$\begin{aligned} Y(k+1) = {} & a_0Y(k) + \,..\, + a_nY(k-n+1) + a_{n+1}g(Y(k)) \\ & +b_0U(k) + \,..\, + b_mU(k-m) \end{aligned}$$

This can be written in terms of a parameter and regression vector

$$\begin{aligned} Y(k+1) &= \underline{\phi}^T\underline{\theta} \\ \underline{\phi}^T &= [a_1, \ldots, a_{n+1}, b_0, \ldots, b_m]. \end{aligned}$$

Using this parameterization, the recursive least squares technique is directly applicable.

The back-substitution technique for reducing the magnitude of the parameter estimation is described in the following subsection for the three-degree-of-freedom planar arm system. This discussion includes the control law formulation for model reference control and nonlinearity feedforward cancellation.

<u>Parameter Estimation</u>. - Our starting point is the equations for the three-link planar arm. These can be rewritten for the purposes of parameter estimation as shown.

Row 1

Row 1

$$\begin{aligned} & D_{11}\ddot\theta_1 + D_{12}\ddot\theta_2 + D_{13}\ddot\theta_3 + D_{14}\left[\cos\theta_2\ (2\ddot\theta_1 + \ddot\theta_2) - \sin\theta_2(\dot\theta_2^2 + 2\dot\theta_1\dot\theta_2)\right] \\ & + D_{15}\left[\cos(\theta_2 + \theta_3)(2\ddot\theta_1 + \ddot\theta_2) - \sin(\theta_2 + \theta_3)(\dot\theta_1 + \dot\theta_2^2 + 2(\dot\theta_1\dot\theta_2 + \dot\theta_2\dot\theta_3 + \dot\theta_1\dot\theta_3))\right] \\ & + D_{16}\left[\cos\theta_3(2\ddot\theta_1 + 2\ddot\theta_2 + \ddot\theta_3) - \sin\theta_3(\dot\theta_3^2 + 2\dot\theta_1\dot\theta_3 + 2\dot\theta_2\dot\theta_3)\right] + D_{17}\dot\theta_1 \\ & + D_{18}\sin\theta_1 + D_{19}\sin(\theta_1 + \theta_2) + D_{1,10}\sin(\theta_1 + \theta_2 + \theta_3) = T_1 \end{aligned}$$

Row 2

$$D_{21}\left[\ddot{\theta}_1 + \ddot{\theta}_2\right] + D_{22}\ddot{\theta}_3 + D_{23}\left[\cos\theta_2(\ddot{\theta}_1) - \sin\theta_2(\dot{\theta}_1^2)\right]$$
$$+ D_{24}\left[\cos(\theta_2 + \theta_3)(\ddot{\theta}_1) - \sin(\theta_2 + \theta_3)(\dot{\theta}_1^2)\right]$$
$$+ D_{25}\left[\cos\theta_3(2\ddot{\theta}_1 + 2\ddot{\theta}_2 + \ddot{\theta}_3) - \sin\theta_3(\dot{\theta}_3^2 + 2\dot{\theta}_1\dot{\theta}_3 + 2\dot{\theta}_2\dot{\theta}_3)\right] + D_{26}\dot{\theta}_2$$
$$+ D_{27}\sin(\theta_1 + \theta_2) + D_{28}\sin(\theta_1 + \theta_2 + \theta_3) = T_2 .$$

Row 3

$$D_{31}\left[\ddot{\theta}_1 + \ddot{\theta}_2 + \ddot{\theta}_3\right] + D_{32}\left[\cos(\theta_2 + \theta_3)(\ddot{\theta}_1) - \sin(\theta_2 + \theta_3)\dot{\theta}_1^2\right]$$
$$+ D_{33}\left[\cos\theta_3(\ddot{\theta}_1 + \ddot{\theta}_2) - \sin\theta_3(\dot{\theta}_1 - \dot{\theta}_2)^2\right] + D_{34}\dot{\theta}_3$$
$$+ D_{35}\sin(\theta_1 + \theta_2 + \theta_3) = T_3 .$$

where

Row 1

$$D_{11} = M_1a_1^2 + (L_1^2 + a_2^2) + M_3(L_1^2 + L_2^2 + a_3^2) + I_1 + I_2 + I_3$$
$$D_{12} = M_2a_2^2 + M_3\ (L_2^2 + a_2^3) + I_2 + I_3$$
$$D_{13} = M_3a_3^2 + I_3$$
$$D_{14} = M_2L_1a_2 + M_3L_1L_2$$
$$D_{15} = M_3L_1a_3$$
$$D_{16} = M_3L_2a_3$$
$$D_{17} = B_1$$
$$D_{18} = M_1a_1g + M_2L_1g + M_3L_1g$$
$$D_{19} = M_2a_2g + M_3L_2g$$
$$D_{1,10} = M_3a_3g .$$

Row 2

$D_{21} = M_2a_2^2 + M_3(L_2^2 + a_3^2) + I_2 + I_3$

$D_{22} = M_3a_3^2 + I_3$

$D_{23} = M_2L_1a_2 + M_3L_1L_2$

$D_{24} = M_3L_1a_3$

$D_{25} = M_3L_2a_3$

$D_{26} = B_1$

$D_{27} = M_2a_2g + M_3L_2g$

$D_{28} = M_3a_3g$

Row 3

$D_{31} = M_3a_3^2 + I_3$

$D_{32} = M_3L_1a_3$

$D_{33} = M_3L_2a_3$

$D_{34} = B_3$

$D_{35} = M_3a_3g$.

If we assume knowledge of L_1, L_2 and g, we can rewrite these equations as

Row 1

$$D_{11}\ddot{\theta}_1 + D_{12}\ddot{\theta}_2 + D_{13}\ddot{\theta}_3 +$$
$$D'_{14}\left[L_1(\cos\theta_2(2\ddot{\theta}_1 + \ddot{\theta}_2) - \sin\theta_2(\dot{\theta}_2^2 + 2\dot{\theta}_1\dot{\theta}_2) + g\sin(\theta_1 + \theta_2)\right]$$
$$D'_{15}\Big[L_1(\cos(\theta_2 + \theta_3)(2\ddot{\theta}_1 + \ddot{\theta}_2 + \ddot{\theta}_3) - \sin(\theta_2 + \theta_3)(\dot{\theta}_3^2 + \dot{\theta}_2^2 + 2(\dot{\theta}_1\dot{\theta}_2 + \dot{\theta}_1\dot{\theta}_3 + \dot{\theta}_2\dot{\theta}_3)))$$
$$+ L_2(\cos\theta_3(2\ddot{\theta}_1 + 2\ddot{\theta}_2 + \ddot{\theta}_3) - \sin\theta_3(\dot{\theta}_3^2 + 2\dot{\theta}_1\dot{\theta}_3 + 2\dot{\theta}_2\dot{\theta}_3))$$
$$+ g\sin(\theta_1 + \theta_2 + \theta_3)\Big]$$
$$+ D'_{17}\dot{\theta}_1 + D'_{18}(g\sin\theta_1) = T_1 .$$

Row 2

$$D_{21}\left[\ddot{\theta}_1 + \ddot{\theta}_2\right] + D_{22}\ddot{\theta}_3 + D'_{23}\left[L_1(\cos\theta_2(\ddot{\theta}_1) + \sin\theta_2(\dot{\theta}_1^2) + g\sin(\theta_1 + \theta_2)\right]$$
$$+ D'_{24}\left[L_1(\cos(\theta_2 + \theta_3)(\ddot{\theta}_1) + \sin\ (\theta_2 + \theta_3)(\dot{\theta}_1^2)) + g\sin(\theta_1 + \theta_2 + \theta_3) + L_2(\cos_3(2\ddot{\theta}_1 + 2\ddot{\theta}_2 + \ddot{\theta}_3) - \sin\theta_3(\dot{\theta}_3^2 + 2\dot{\theta}_1\dot{\theta}_3 + 2\dot{\theta}_3\dot{\theta}_2))\right]$$
$$+ D_{26}\dot{\theta}_2 = T_2$$

Row 3

$$D_{31}\left[\ddot{\theta}_1 + \ddot{\theta}_2 + \ddot{\theta}_3\right]$$
$$+ D'_{32}\left[L_1(\cos(\theta_2 + \theta_3)(\ddot{\theta}_1) + \sin(\theta_2 + \theta_3)\dot{\theta}_1^2) + g\sin(\theta_1 + \theta_2 + \theta_3) + L_2(\cos\theta_3(\ddot{\theta}_1 + \ddot{\theta}_2) + \sin\theta_3(\dot{\theta}_1^2 + 2\dot{\theta}_1\dot{\theta}_2 + \dot{\theta}_2^2))\right]$$
$$+ D_{34}\dot{\theta}_3 = T_3$$

where

$D'_{14} = M_2a_2 + M_3L_2$	$D'_{23} = M_2a_2 + M_3L_2$
$D'_{15} = M_3a_3$	$D'_{24} = M_3a_3$
$D'_{18} = M_1a_1 + M_2L_1 + M_3L_1$	$D'_{32} = M_3a_3$

This reduces the number of parameters from 10, 8, and 5 for rows 1, 2, and 3 to 7, 5, and 3. Parameter estimation of 7, 5, and 3 parameters is very reasonable and could be implemented practically. However, it is possible to further simplify the size of the parameter estimation problems for rows 1 and 2 by back-substituting estimates from rows 3 and 2.

Notice that

$$D'_{24} = D'_{32}$$

$$D_{22} = D_{31}$$

Thus the row 2 equation can be written as

$$D_{21}\left[\ddot{\theta}_1 + \ddot{\theta}_2\right] + D'_{23}\left[L_1(\cos\theta_2(\ddot{\theta}_1) + \sin\theta_2(\dot{\theta}_1^2) + g\sin(\theta_1 + \theta_2)\right] + D_{26}\dot{\theta}_2$$

$$= T_2 - \hat{D}_{31}\ddot{\theta}_3 - \hat{D}'_{32}\ L_1(\cos(\theta_2 + \theta_3)(\ddot{\theta}_1) + \sin(\theta_2 + \theta_3)(\dot{\theta}_1^2))$$

$$+ L_2(\cos\theta_3(2\ddot{\theta}_1 + 2\ddot{\theta}_2 + \ddot{\theta}_3) - \sin\theta_3(\dot{\theta}_3^2 + 2\theta_1\theta_3 + 2\dot{\theta}_2\dot{\theta}_3)$$

$$+ g\sin(\theta_1 + \theta_2 + \theta_3)$$

where

$\hat{D}_{31}$ = estimate of D_{31}

$\hat{D}'_{32}$ = estimate of D'_{32}.

Similarly by observing

$D_{12} = D_{21}$

$D_{13} = D_{31}$

$D'_{15} = D'_{32}$

We can rewrite the equation for row 1 as

$$D_{11}\ddot{\theta}_1 + D_{17}\dot{\theta}_1 + D_{18}\left[g\sin\theta_3\right] = T_1 - \hat{D}_{31}\ddot{\theta}_3 - \hat{D}_{21}\ddot{\theta}_2$$

$$- \hat{D}'_{23}\left[L_1(\cos\theta_2(2\ddot{\theta}_1 + \ddot{\theta}_2) - \sin\theta_2(\dot{\theta}_2^2 + 2\dot{\theta}_1\dot{\theta}_2)) + g\sin(\theta_1 + \theta_2)\right]$$

$$- D'_{32}\Big[L_1(\cos(\theta_2 + \theta_3)(2\ddot{\theta}_1 + \ddot{\theta}_2 + \ddot{\theta}_3)$$

$$- \sin(\theta_2 + \theta_3)(\dot{\theta}_3^2 + \dot{\theta}_2^2 + 2(\dot{\theta}_1\dot{\theta}_2 + \dot{\theta}_1\dot{\theta}_3 + \dot{\theta}_2\dot{\theta}_3))$$

$$+ L_2(\cos\theta_3(2\ddot{\theta}_1 + 2\ddot{\theta}_2 + \ddot{\theta}_3) - \sin\theta_3(\dot{\theta}_3^2 + 2\dot{\theta}_1\dot{\theta}_3 + 2\dot{\theta}_2\dot{\theta}_3))$$

$$+ g\sin(\theta_1 + \theta_2 + \theta_3)\Big].$$

Each of these equations is now in the form

$$\bar{X}_i^T\ \bar{\Theta}_i = Y_i$$

where

$\underline{X}_i$ and Y_i

contain measurable signals and Θ is a vector of parameters to be estimated, i.e.,

$$\Theta_1 = [D_{11}, D_{17}, D_{18}]$$
$$\Theta_2 = [D_{21}, D_{23}, D_{26}]$$
$$\Theta_3 = [D_{31}, D_{32}, D_{34}]$$

To generate X and Y, we approximate $\ddot{\theta}_i$ and $\dot{\theta}_i$ as

$$\ddot{\theta}_{ik} = \ddot{\theta}_i(kh) = \frac{\theta_i((k+1)h) - 2\theta_i(kh) + \theta_i((k-1)h)}{h^2} \triangleq A_{ik}$$

$$\dot{\theta}_{ik} = \dot{\theta}_i(kh) = \frac{\theta_i(kh) - \theta_i((k-1)h)}{h} \triangleq V_{ik}$$

Notice that one added benefit of this approach is that now all parameters in Θ_i are of the same order of magnitude. This helps the performance of sequential least squares.

Model matching control. - Assume, based on the equations for the three-link planar arm, the following discretized model of the planar manipulator

$$J(\theta_k) A_k + BV_k + N_k = T_k \qquad h = \text{sample period}$$

where $A_{ik} = \frac{1}{h^2}(\theta_{ik+1} - 2\theta_{ik} + \theta_{ik-1})$

$$V_k = \begin{bmatrix} V_{1k} \\ V_{2k} \\ V_{3k} \end{bmatrix} \quad A_k = \begin{bmatrix} A_{1k} \\ A_{2k} \\ A_{3k} \end{bmatrix} \quad \theta_k = \begin{bmatrix} \theta_{1k} \\ \theta_{2k} \\ \theta_{3k} \end{bmatrix} \quad T_k = \begin{bmatrix} T_{1k} \\ T_{2k} \\ T_{3k} \end{bmatrix}$$

$$V_{ik} = \frac{1}{h}(\theta_{ik} - \theta_{ik-1}) \qquad B = \text{diag}\,(D_{17}, D_{26}, D_{34})$$

$$J(\theta_k) = \begin{bmatrix} \begin{pmatrix} D_{11} + 2D_{14}\cos\theta_{2_k} \\ + 2D_{15}\cos(\theta_{2_k} + \theta_{3_k}) \\ + 2D_{16}\cos\theta_{3_k} \end{pmatrix} & \begin{pmatrix} D_{12} + D_{14}\cos\theta_{2_k} \\ + D_{15}\cos(\theta_{2_k} + \theta_{3_k}) \\ + 2D_{16}\cos\theta_{3_k} \end{pmatrix} & \begin{pmatrix} D_{13} \\ + D_{16}\cos\theta_{3_k} \end{pmatrix} \\ \begin{pmatrix} D_{21} + D_{23}\cos\theta_{2_k} \\ + D_{24}\cos(\theta_{2_k} + \theta_{3_k}) \\ + 2D_{25}\cos\theta_{3_k} \end{pmatrix} & \begin{pmatrix} D_{21} \\ + 2D_{25}\cos\theta_{3_k} \end{pmatrix} & \begin{pmatrix} D_{22} \\ + D_{25}\cos\theta_{3_k} \end{pmatrix} \\ \begin{pmatrix} D_{31} \\ + D_{32}\cos(\theta_2 + \theta_{3_k}) \\ + D_{33}\cos\theta_{3_k} \end{pmatrix} & \begin{pmatrix} D_{31} \\ + D_{33}\cos\theta_{3_k} \end{pmatrix} & \begin{pmatrix} D_{31} \end{pmatrix} \end{bmatrix}$$

$$N_k = \begin{bmatrix} N_{1_k} \\ N_{2_k} \\ N_{3_k} \end{bmatrix}$$

$$\begin{aligned} N_{1_k} &= -D_{14}\left[\sin\theta_{2_k}(V_{2_k}^2 + 2V_{1_k}V_{2_k})\right] - D_{15}\Big[\sin(\theta_{2_k} + \theta_{3_k})(V_{1_k}^2 + V_{2_k}^2 + \\ &\quad 2(V_{1_k}V_{2_k} + V_{2_k}V_{3_k} + V_{1_k}V_{3_k})\Big] - D_{16}\left[\sin\theta_{3_k}(V_{3_k}^2 + 2V_{1_k}V_{3_k} + 2V_{2_k}V_{3_k})\right] \\ N_{2_k} &= -D_{23}\left[\sin\theta_{2_k}(V_{1_k}^2)\right] - D_{24}\left[\sin(\theta_{2_k} + \theta_{3_k})V_{1_k}^2\right] \\ &\quad + D_{25}\left[-\sin\theta_{3_k}(V_{3_k}^2 + 2(V_{1_k}V_{3_k} + V_{2_k}V_{3_k}))\right] \\ N_{3_k} &= -D_{32}\left[\sin(\theta_{2_k} + \theta_{3_k})V_{1_k}^2\right] - D_{33}\left[\sin\theta_3(V_{1_k} - V_{2_k})^2\right]. \end{aligned}$$

Assume we want the closed-loop system to be characterized by the model

$$\begin{aligned} E_{11}\theta_{1_{k+1}} + E_{12}\theta_{1_k} + E_{13}\theta_{1_{k-1}} &= H_{11}\Gamma_{1_k} + H_{12}\Gamma_{1_{k-1}} \\ E_{21}\theta_{2_{k+1}} + E_{22}\theta_{2_k} + E_{23}\theta_{2_{k-1}} &= H_{21}\Gamma_{2_k} + H_{22}\Gamma_{2_{k-1}} \\ E_{31}\theta_{2_{k+1}} + E_{32}\theta_{2_k} + E_{33}\theta_{2_{k-1}} &= H_{31}\Gamma_{3_k} + H_{32}\Gamma_{3_{k-1}} \end{aligned}$$

An appropriate control is simply

$$\underline{T_k} = \underline{N_k} + \underline{BV_k} + \underline{J(\theta_k)W_k}$$

$$\underline{W_k} = \frac{1}{h^2} \begin{bmatrix} W_{k1} \\ W_{k2} \\ W_{k3} \end{bmatrix}$$

$$W_{ki} = -\left[E_{12} + 2\right]\theta_{ik} + \left[E_{13} + 1\right]\theta_{i(k-1)}$$

$$+ H_{i1}\Gamma_{ik} + H_{i2}\Gamma_{i(k-1)}$$

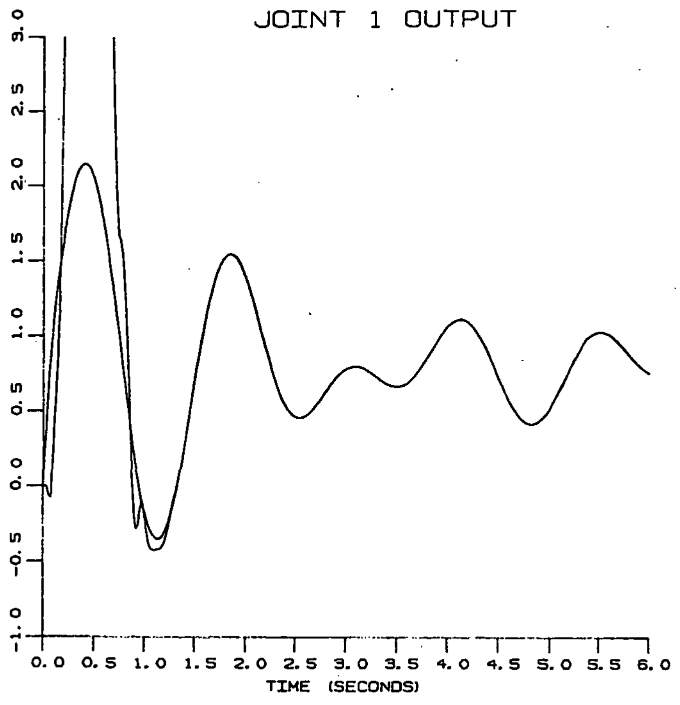
An adaptive version of this fixed control is obtained by generating estimates of

$J(\theta_k)$, B, and N

using the estimates of $\hat{\Theta}_1$ and the equations relating $\hat{\Theta}_1$ to J, B and N derived earlier.

Simulation results for a three-degree-of-freedom planar arm system are shown in Figure 36(a) thru (c). For each joint, both tracking performance and parameter estimation results are shown

A full implementation of the adaptive control strategy described for a 6-DOF or 7-DOF system would represent a significant computational burden. There are, however, many ways to reduce the overall complexity of the problem. The most obvious approach for reducing the severity of the problem is to decrease the complexity of the underlying dynamic equations by ignoring certain terms, specifically centrifugal and coriolis acceleration. A second approach to making the problem more tractable is to treat the manipulator as two decoupled systems. Both approaches will be investigated as future research topics.



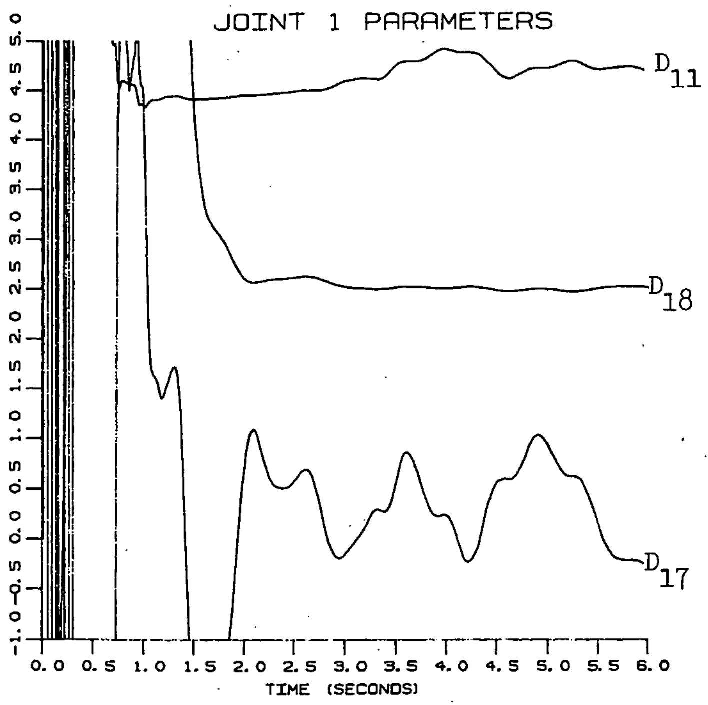
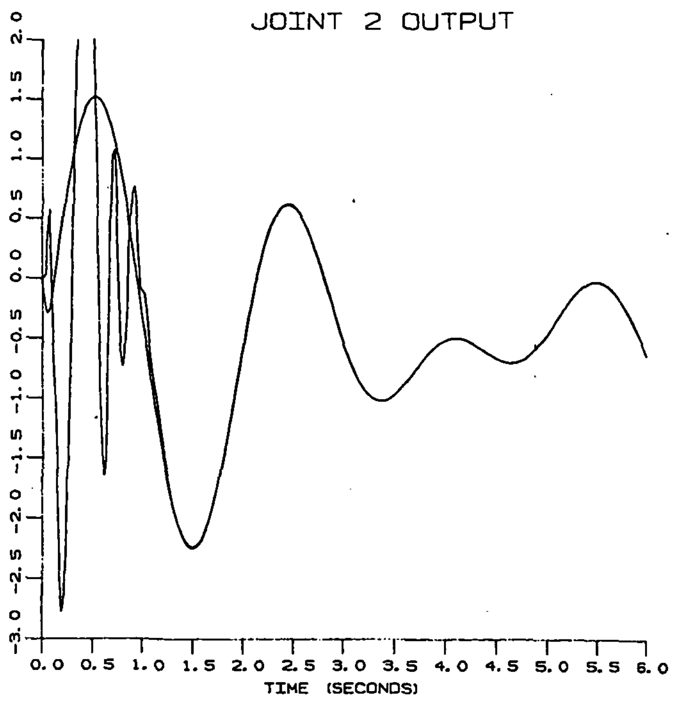


Figure 36(a). - 3-DOF adaptive control simulation results - joint 1.



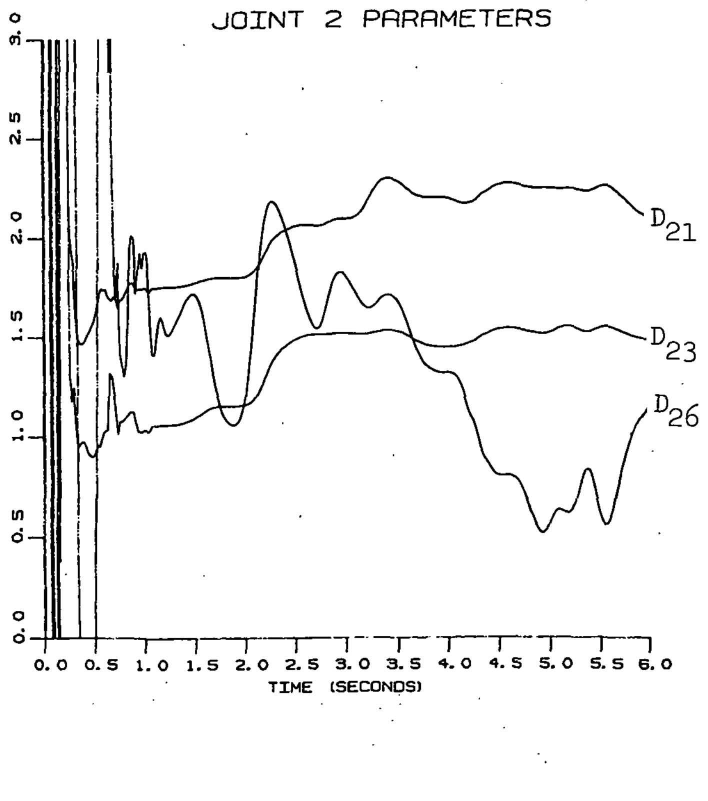
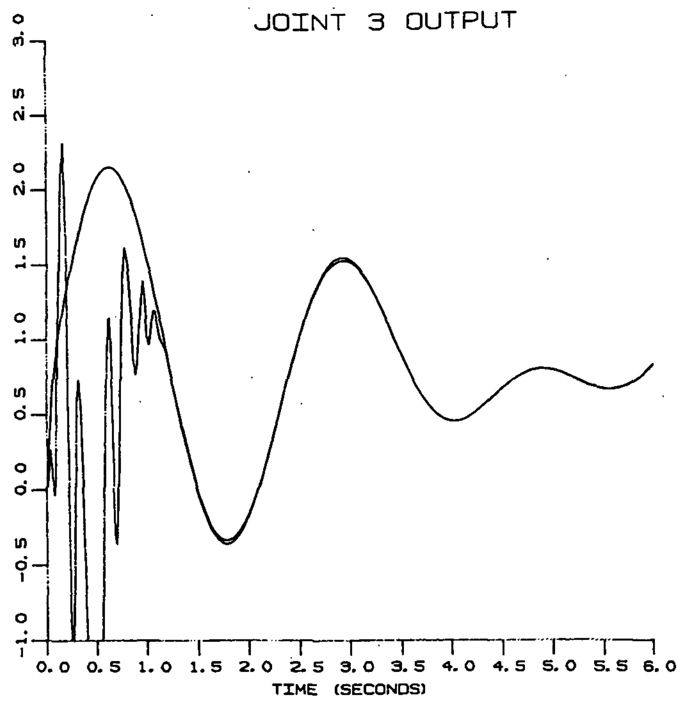


Figure 36(b). - 3-DOF adaptive control simulation results - joint 2.



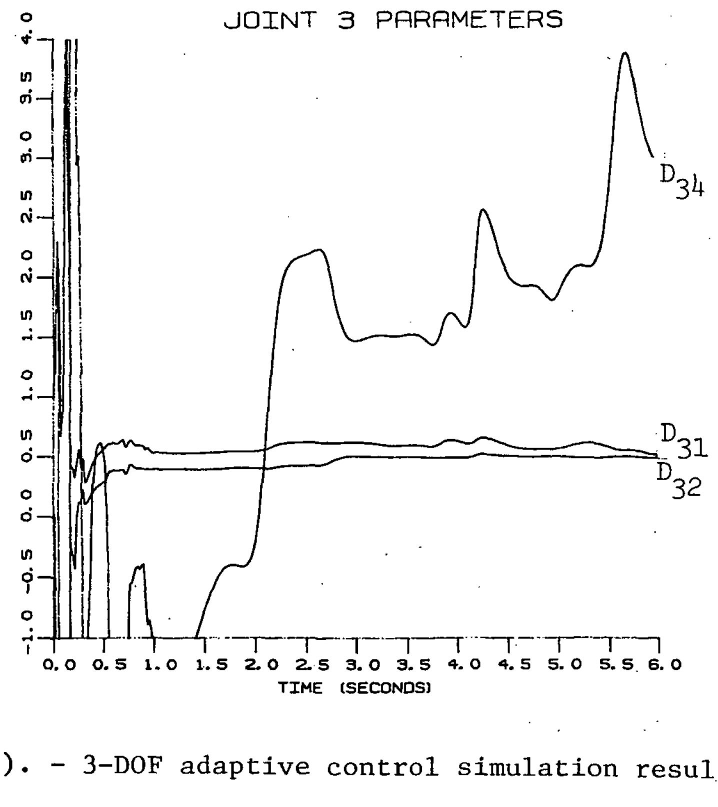


Figure 36(c). - 3-DOF adaptive control simulation results - joint 3.

The previous sections have discussed manipulator control in the context of pure positional control. The topic of this section is the control for tasks that involve satisfying both positional and force/torque constraints. For the remainder of the section, this type of control will be referred to as "force/torque" control.

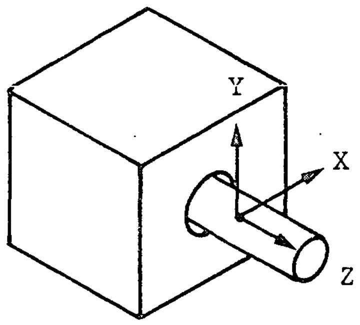
Perhaps the most common example of a task that requires simultaneous satisfaction of position and force/torque constraints is the insertion of a peg in a hole. This example will be used to introduce a convention for describing tasks that was suggested in [Mason 1981]. The peg-in-the-hole physical model is shown in Figure 37.

<u>Natural Constraints</u>

$V_x = 0 \qquad W_y = 0$

$V_y = 0 \qquad F_z = 0$

$W_x = 0 \qquad T_z = 0$



<u>Artificial Constraints</u>

$F_x = F_y = T_x = T_y = 0, V_z = \alpha, W_z = 0$

Figure 37. - Compliant control for peg insertion.

Using Mason's convention, a task is described in terms of "natural" and "artificial" constraints specified in a coordinate frame associated with the task. Natural constraints are those associated with the task physical geometry. In the case of the peg-in-the-hole, for example, motion in the X and Y axes appears as a natural constraint because it is constrained by the sides of the hole. Similarly, forces and torques about the Z-axis appear as natural constraints (assuming frictionless surfaces between the peg and hole). Artificial constraints essentially determine how the task will be performed. Again referring to the example, the artificial constraints include centering of the peg (X and Y axes forces and torques equal to zero), and motion of the peg along the Z-axis ($V_Z =$).

The two fundamental approaches to force/torque control are generally referred to as passive and active compliance. Passively compliant techniques rely on the use of a mechanically compliant device located near the end-effector. Misalignments at the workpiece are compensated for by the compliant action of this device. The basic idea is shown in Figure 38 taken from [Whitney 1982]. While passive compliance can be used for many tasks, the approach has certain drawbacks. The most significant is the mechanical resonance introduced by the passive device.

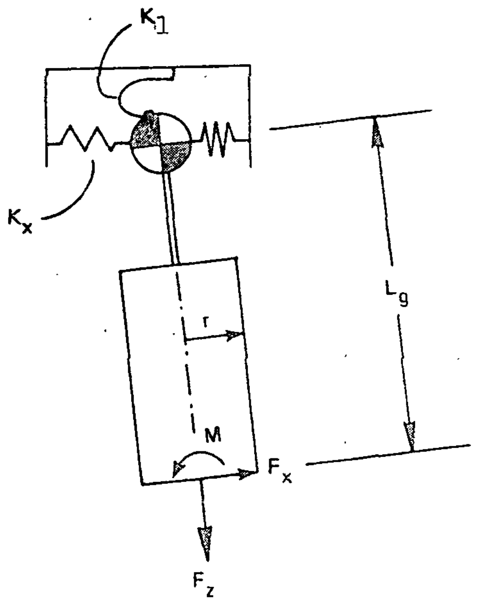


Figure 38. - Rigid peg supported compliantly by lateral spring K_x and angular spring K_1 at distance L_g from peg's tip.

The alternative to passively compliant devices is an active compliance strategy. In an active compliance approach the forces and torques at the end-effector are measured and this information is used in a feedback loop to provide servoforce control.

The forces and torques at the end-effector are related to joint torques by the Jacobian transpose

$$\underline{T} = [J]^T\underline{F}$$

where

$\underline{T}$ = Joint torques
$\underline{F}$ = End effector forces and torques
$[J]$ - Jacobian matrix

This relationship is used to map force error commands from an external reference frame to the manipulator joints. It also demonstrates two alternatives for measuring end-effector forces and torques. First, joint torques can be used to reconstruct end-effector forces and torques via the Jacobian relationship. Joint torques can be measured in a variety of ways. First, they can be measured directly via strain gage measurement devices [Paul 1981a]. In another approach, joint torques can be derived from joint motor currents through the motor torque constants.

A second approach to generating measurements of end-effector forces and torques is to use a wrist-mounted sensing device. While these devices are becoming more readily available from a number of manufacturers, the most popular implementation is shown in Figure 39. This is generally referred to as a "Scheinman wrist", (after its inventor), and uses 16 strain gages to resolve forces and torques. Strain-gage-type force-sensing devices typically have a force resolution of 3 to 5 oz.

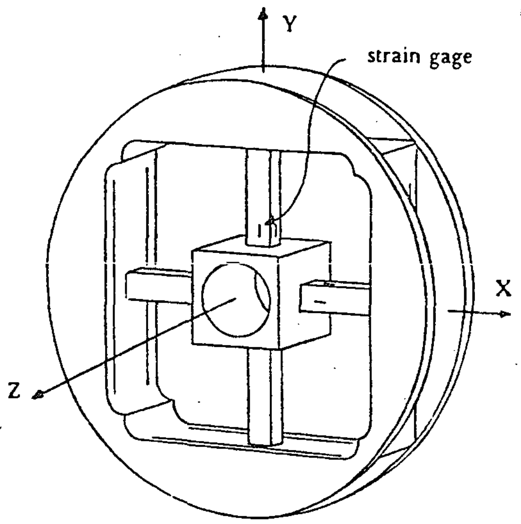


Figure 39. - Scheinman force-sensing wrist.

The force-sensing wrist approach has two key advantages over the joint torque measurement approach for determining end-effector forces and torques. The first advantage is that the accuracy of the joint torque sensing approach is degraded by friction, both viscous and non-linear that appears in all manipulators. Also, the derivation of forces and torques is more computationally intensive for the joint torque approach [Shimano 1979].

Two approaches to force/torque control will be discussed. The first approach was introduced by Raibert and Craig while at JPL, [Raibert 1981]. This strategy involves the use of separate position and force servo loops. Errors are generated in a Cartesian reference frame and then mapped into the joint reference frame. The response torque at each joint aids in satisfying both position and force objectives in the Cartesian space. The approach is referred to by the authors as "hybrid" control. The second approach was introduced in [Salisbury 1980], and has been coined "active stiffness" control. The controller has the net result of causing the end-effector of the manipulator to behave as a six-degree-of-freedom spring in space. This is very similar to the passively compliant techniques described earlier with the important exception that stiffnesses are computer controlled and can be varied as a function of the particular task being performed.

Hybrid Control. - As introduced in the previous section, the hybrid control approach is based on the description of a manipulator task in terms of position- and force-controlled axes in a task reference coordinate frame. The natural constraints partition the degrees of freedom into a position-controlled subset and a force-controlled subset. The desired position and force trajectories are then specified by the artificial constraints. Figure 40 illustrates the basic structure of the hybrid controller. This figure will be used to describe additional detail of controller functioning.

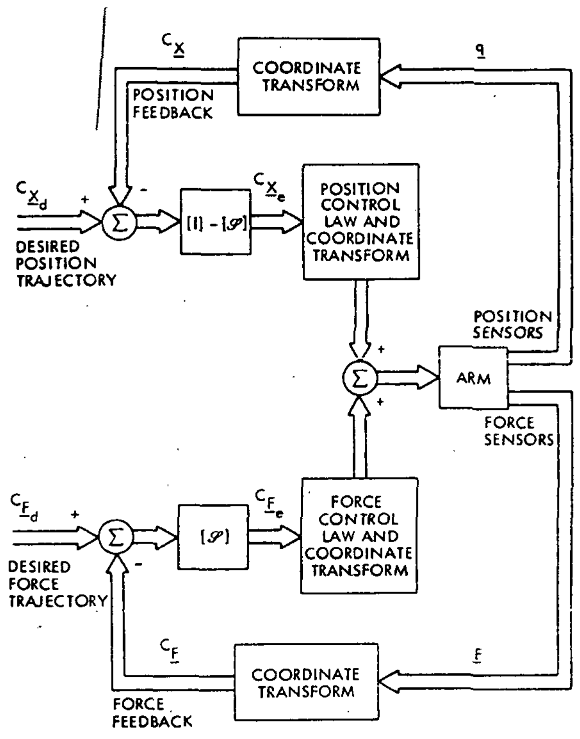


Figure 40. - Conceptual organization of hybrid controller.

Error signals are generated in the task reference frame by comparing desired trajectories with the actual state of the manipulator. This involves two comparisons, as shown in Figure 40. To create the position error (the upper loop in Fig. 40), manipulator position in the task coordinate system is derived from a kinematic transformation of the manipulator joint angles and subtracted from the reference trajectory. A force error is derived in a similar manner as shown in the lower loop of Figure 40. This results in a force feedback vector that is compared with the desired force trajectory to generate a force error vector.

The next step in the hybrid control process is "filtering" of the position and force error signals by a "compliance selection matrix" [S]. The compliance selection matrix is required because the transformations of manipulator position and force variables result in terms that violate the partitioning of the task reference frame into orthogonal position and force axes. The compliance selection matrix has the basic form:

$$S = \begin{bmatrix} S_1 & & & & & \\ & S_2 & & -0- & & \\ & & S_3 & & & \\ & & & S_4 & & \\ -0- & & & & S_5 & \\ & & & & & S_6 \end{bmatrix}$$

$S_i = 1$ if DOF X_i is force controlled, $S_i = 0$ otherwise

When the force error vector is mapped through the compliance selection matrix, force components sensed at the manipulator end-effector (in general, components will exist on all six axes) that are not on the force-controlled DOF will be eliminated from the error vector. A similar operation occurs for the position-controlled DOF; however in this case the error vector is mapped through [I] - [S], where [I] is the identity matrix.

In the final stage of the hybrid control process, the error vectors for both force and position DOF are mapped back into manipulator joint space as shown below.

$$\underline{q}e(t) = [J]^{-1}{}^{C}\underline{X}e(t)$$

$$\underline{\tau}e(t) = [J]^{T}{}^{C}\underline{F}e(t)$$

$\underline{X}e(t)$:	Position error vector in task coordinate
$\underline{F}e(t)$:	Force error vector in task coordinate
$\underline{q}e(t)$:	Position error vector in joint coordinate
$\underline{\tau}e(t)$:	Torque error vector in joint coordinate
[J]:	Manipulator Jacobian

The error vectors, $\underline{q}$ and $\underline{\tau}$, are then distributed to the individual joint servos of the manipulator. A more extensive diagram of the servo control structure proposed in [Raibert 1981] is shown in Figure 41.

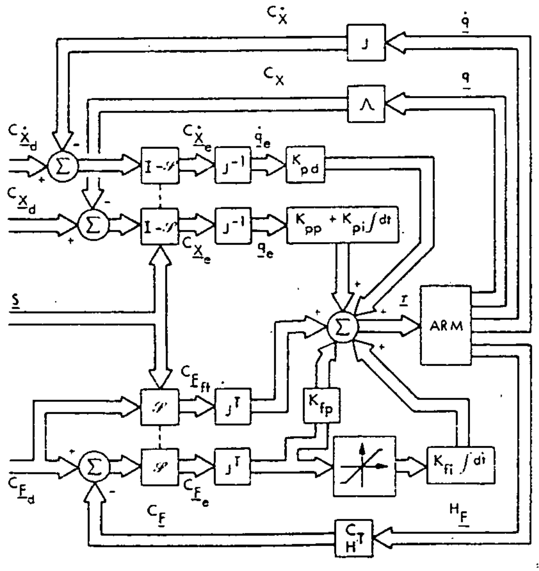


Figure 41. - Hybrid controller implementation.

The basic hybrid control approach described above has been implemented on ROBSIM for a three-degree-of-freedom planar arm case. A sketch of the simulation configuration is shown in Figure 42.

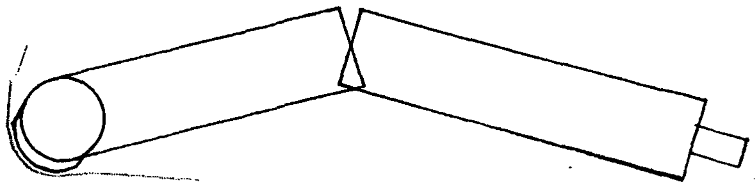


Figure 42. - Hybrid control simulation configuration.

The task consisted of applying a force of 10 newtons in the Z-axis while maintaining a specified Y-position. A plot of the resulting force profile is shown in Figure 43.

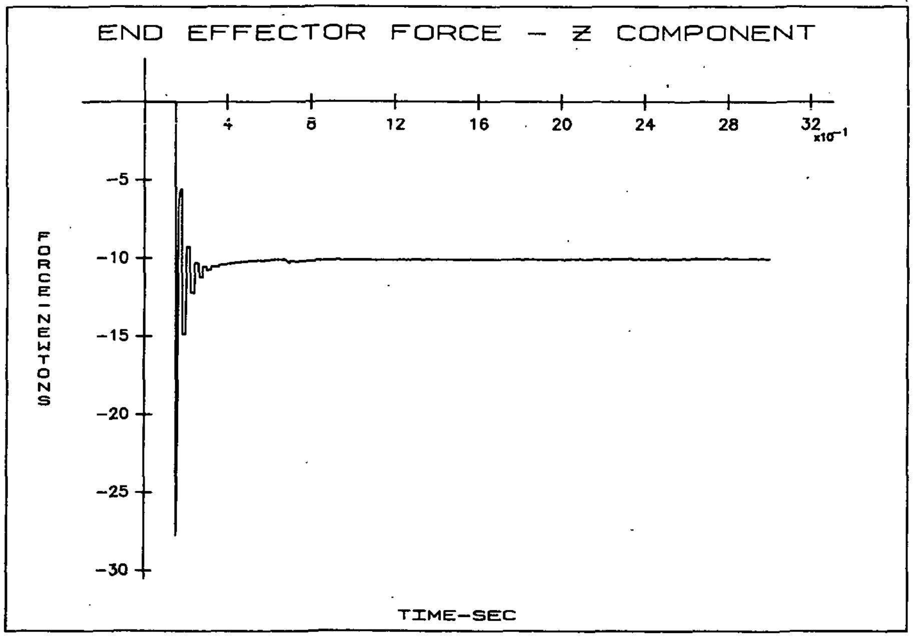


Figure 43. - Profile of controlled force component.

Active stiffness control. - The concept of active stiffness control has many of the attributes of the hybrid control strategy described earlier, but introduces the novel concept of forcing the end-effector of the manipulator to behave as a six-dimensional spring in the Cartesian reference frame. As pointed out earlier, this bears a striking resemblance to passive compliance approaches in terms of accommodating misalignments between the end effector and task. Again, the important difference is that in an active compliance approach, stiffnesses on specific axes are controlled externally and varied during different phases of a task. For example, during rapid slewing motions in moving from point to point, stiffnesses can be set high to minimize reactions to acceleration-induced forces from the load or tool.

To arrive at an equivalent six-dimensional spring relationship expressed in terms of joint displacements and joint torques

$$T = [J]^T K [J] \delta \underline{\theta}$$

The matrix $K_\theta = J^T K J$ is referred to as the joint stiffness matrix. The point in the hand reference frame that serves as the basis for the computation of the Jacobian is referred to as the stiffness center. This is the point at which pure forces cause only translation and pure torques cause only rotation.

In its most basic form active stiffness control can be implemented by generating joint torque commands of the form

$$\underline{T}_C = K_\theta \, \delta\underline{\theta} + \underline{T}_B$$

The term $\underline{T}_B$ is a bias torque generated from the relationship $\underline{T}_B = J^T \underline{F}_B$. $\underline{F}_B$ represents bias forces in the Cartesian reference frame. Note that the appropriate choice of K leads to a decomposition of force- and position-controlled axes as in the hybrid control scheme.

A six-dimensional linear spring is described by the relationship

$$\underline{F} = [K] \delta \underline{X}$$

$\delta \underline{X}$ is a generalized displacement from a nominal position $\underline{X}_0$ of the hand origin and consists of three orthogonal translation components and three small rotations about orthogonal axes. This diagonal matrix [K] contains the spring coefficients for each axis.

The Cartesian displacement $\delta \underline{X}$ can be related to a manipulator joint displacement $\delta \underline{\theta}$, ($\delta \underline{\theta} = \underline{\theta} - \underline{\theta}_0$) through the Jacobian matrix

$$\delta \underline{X} = [J] \delta \underline{\theta}$$

Joint torques are also related to end-effector forces and torques as a function of the Jacobian

$$\underline{T} = [J]^T \underline{F}$$

These expressions can be combined to obtain the stiffness matrix K_θ.

In actual implementation of active stiffness control, additional compensation elements were required [Salisbury 1980]. The total torque applied to the ith joint is given by

$$T_i = T_{C,i} + G_i \delta T_i + K_{V,i} C_{II,i} \delta \dot{\theta}_i + V_{0,i} sgn(\dot{\theta}_i) + C_{I,i}$$

where:

$T_{C,i}$ =commanded torque, ith joint
δT_i =torque error, ith joint
$\dot{\theta}_i$ =velocity, ith joint
$\delta \dot{\theta}_i$ =velocity error, ith joint
G_i=torque compensation function, ith joint
$K_{V,i}$ =velocity damping term, ith joint
$C_{II,i}$ =instantaneous inertia, ith joint
$C_{I,i}$ =gravity loading, ith joint
$V_{0,i}$ =friction torque, ith joint.

A block diagram of the full control loop is shown in Figure 44.

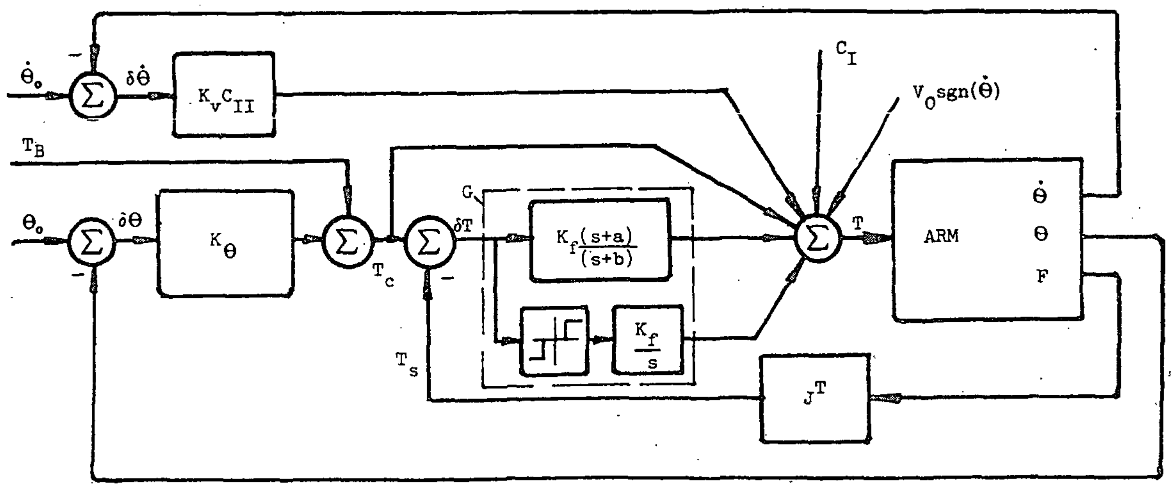


Figure 44. - Stiffness control system.

The algorithm implemented within ROBSIM deviates slightly from that just presented. The first step in this algorithm is the calculation of the end-effector position error $\Delta \underline{P}$. This error vector and any prescribed bias forces $\underline{F}_B$ are used to compute joint control torques

$$\underline{T}_d = [J]^T[K] \ \underline{P} + [J]^T\underline{F}_B$$

where [J] and [K] have the same definitions as stated earlier. A torque error vector $\Delta\underline{T}$ is then determined from

$$\Delta\underline{T} = \underline{T}_c - [J]^T\underline{F}$$

where $\underline{F}$ is the vector of measured end-effector forces and torques.

The error vector $\underline{T}$ is then calculated using

$$\underline{T} = K_{LL}\left(\frac{s+a}{s+b}\right)\Delta\underline{T} + \frac{K_I\Delta\underline{T}}{s} + T_c - K_d s\underline{\theta}$$

where

K_{LL} = lead-lag filter gain

K_I = integrator gain

K_d = derivative gain

Figure 45 shows a diagram of the control loop implemented in ROBSIM to simulate active stiffness control.

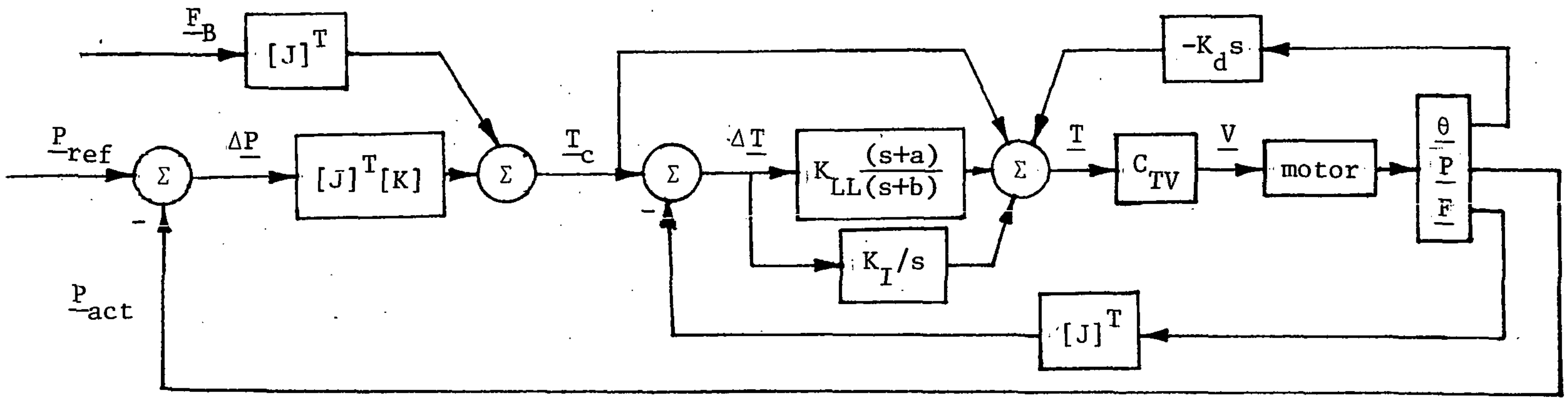


Figure 45. - Stiffness control system implemented in ROBSIM.

The results for an active compliance control simulation are shown for a single axis in Figure 46. In this case, the manipulator has been commanded to a position that is past the boundary of a rigid constraint. The force profile shown is the result of the complying action.

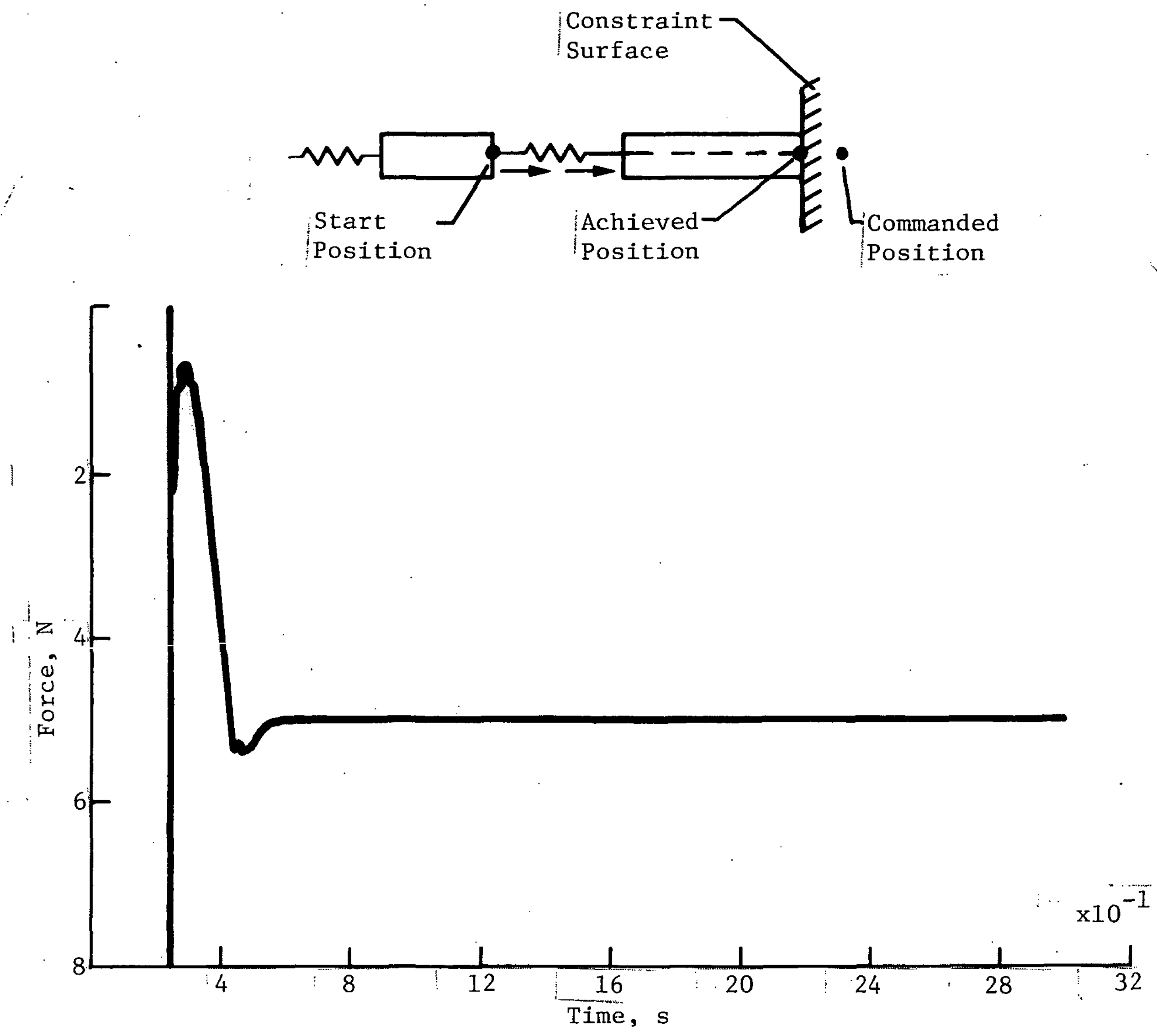


Figure 46. - Active compliance control.

MANIPULATOR TRAJECTORY PLANNING

Manipulator systems often must operate in environments "cluttered" with obstacles, e.g., the objects being manipulated and the machinery with which the arms are interfacing. When performing tasks, the motions of the arms must be carefully planned to avoid collisions between the manipulator links and these obstacles. Collisions between the various parts of the manipulator system must also be prevented, especially in multiple-arm systems or when manipulating bulky objects. Manipulator path-planning can also be used to improve task performance by finding paths that minimize energy consumption or operation time, or that avoid singular configurations of the arm, etc.

To date, path-planning research has focused on two application areas--paths for polyhedral objects and paths for robotic manipulators. Polyhedral objects are self-contained--mobile objects such as autonomous robotic vehicles,* aircraft, etc. The obvious difference between such an object and a manipulator is that the latter is not "mobile" in the sense that one end is fixed to a platform or base that may be mobile. A manipulator arm is a combination of connected links, each of which is a polyhedron. The "motion" of a manipulator is constrained by the link connectivity, which limits its overall reach. The volume (or area) reachable by the hand or tool is a manipulator's working envelope. When a manipulator moves, each of its links sweeps out the path of a polyhedron. Thus, path planning for the motion of polyhedral objects is a subproblem in trajectory planning for manipulators.

The problem of planning paths for polyhedrals has received much attention, with the intent of generalizing methods to manipulator path planning. Unfortunately, the two problems have inherent differences, limiting the application of polyhedral methods to the manipulator case. There have been implementations of manipulator path planners relying solely on relations among polyhedral objects and the polyhedral components of the arm [Brooks 1983b] but these neither capitalize on the serial nature of the manipulator arm nor provide for diverse manipulator operations. They have also demonstrated success only in very simple environments composed of rectangular polyhedrals.

This section of the report discusses the current state of path-planner technology, with the emphasis on path planning for robotic manipulators. The first part consists of the history and development of path-planning methods for both mobile objects and manipulators. The next subsection examines current technology and describes the results of path-planning research under the ROBSIM contract. Finally, the third part compares the current methods and provides some topics for further study. The emphasis in this work is on path planning to avoid obstacles in the manipulator's operating environment. Table IV summarizes the notation employed in this section.

* SRI's Shakey and the Mars rover are two examples of such vehicles.

TABLE IV. - NOTATION FOR MANIPULATOR PATH PLANNING SECTION

d	distance between link and obstacle
$[J_A]$	Jacobian relating translational motion of point A to joint motions
$[J_p]$	Jacobian relating hand motion (translational and rotational) to joint motions
$\underline{n}$	unit vector normal to obstacle surface
$\underline{r}_A$	location (world coordinates) of point A
$\Delta \underline{r}_p$	difference between desired and actual position of hand reference point
ϵ	margin of safety used in preventing collisions
$\Delta \underline{\phi}_p$	vector representing difference between desired and actual hand orientation
θ	joint displacements
$\Delta \theta$	incremental step in joint displacements

Several approaches for solving the path-planning problem for polyhedral objects have been applied. These fall into three categories:

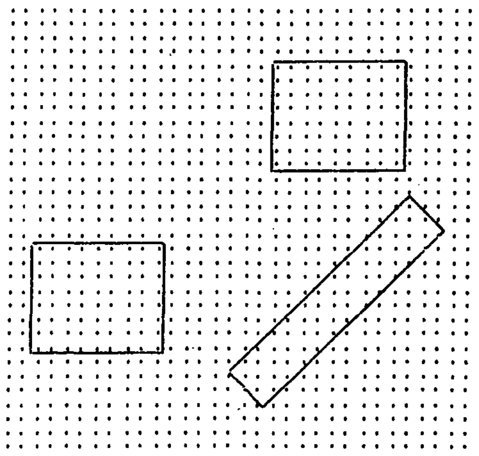
1) Topological search of discretized environments;

2) Visibility lines and growing obstacles;

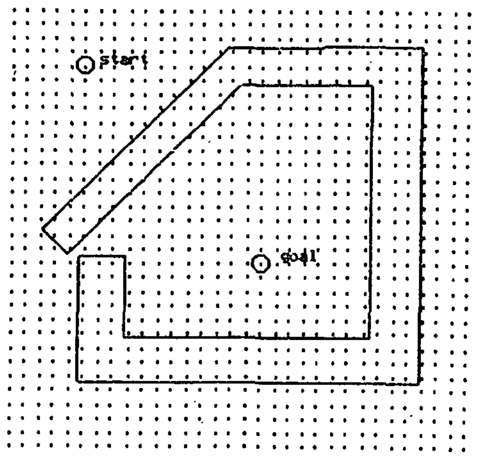
3) Graph search of representations of free space.

Each of these methods has been demonstrated successfully for two-dimensional environments. Some have been generalized to three dimensions with limited success. Each method is discussed in the following paragraphs.

Graph search of discretized environments. - A discretized environment is one in which continuous space has been divided into a set of points, labeled either "free" or "full" depending on whether an object coincides with that point. Searching such a space involves finding a set of connected points ("connected" in the sense that each point in the set lies adjacent to both its predecessor and successor points in the world) from the start point to the goal. There are two basic limitations to this approach. First, both the start and goal positions must be points in the discretized space. If they are not, some other technique must be used to find a path from the start to a nearby point in the space, and from a discrete point to the goal. The second limitation is that a path may not always be found when one exists. Again this is attributable to discretization of the space [Fig. 47(b)]. If the resolution of the discrete representation is not high enough, some free space may not be attainable. Also, small obstacles lying between discrete points may be overlooked. Thus, a discretized space may not allow generation of collision-free paths in all cases. One method for ensuring that objects are not overlooked is to determine the resolution necessary for an environment after considering the size of its objects. Eliminating the problem of nonattainable free space is more difficult.



(a) Discretized space.



(b) Discrete space with impossible goal.

Figure 47.- Discretization of the environment.

Visibility lines and growing obstacles. - A "visibility line" is a path from one point to another in continuous space so that, from either point, one would be able to "see" the other point along an unobstructed straight line. In two dimensions, this approach finds paths using vertices of the objects in the world [Fig. 48(a)]. The basic operation loop of this method is:

1) Propose a straight-line path from current start to current goal;

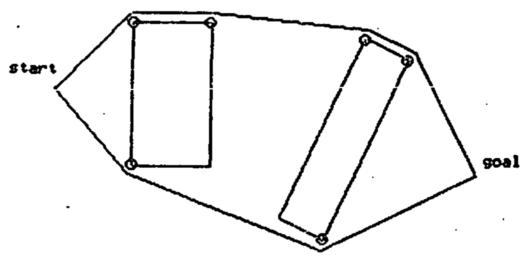
2) If no objects interfere with this path, then return with success; otherwise, go to step 3);

3) Generate subgoal locations avoiding the obstacle;

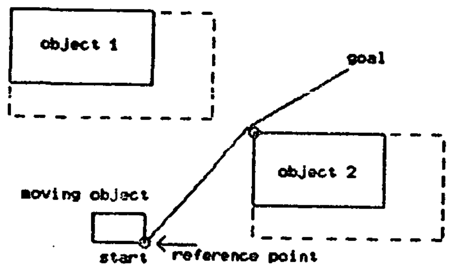
4) Select a new subgoal point to achieve;

5) Go to step 1) recursively.

The algorithm generates a straight-line "visibility" path from one point to another and checks to see if it is permissible. If it is, the path is taken. Otherwise, a path that avoids the obstructing object and attains the goal must be found. The naturally recursive form of this approach makes it attractive for solving the polyhedral path-planning problem.



(a) Visibility line paths.

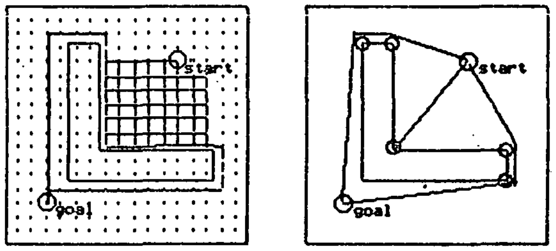


(b) Growing objects.

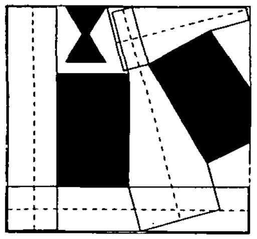
Figure 48.- Visibility line method of path planning.

A severe limitation of this approach is that the object being moved through the environment is considered as a point. This is not so severe when the objects in the environment are many times larger than the moving object. However, some other technique must be used if the moving object is of non-trivial size in comparison with other objects. One method of ensuring collision-free paths in this case is to "grow" the object to be moved [Lozano-Perez 1979]. This technique uses the maximal size of a moving object in a given attitude, a reference point on the object (the point to be moved through the space), and a proposed trajectory to generate areas the reference point must avoid while in motion [Fig. 48(b)]. The simplest case of obstacle growing occurs when the moving obstacle is a circle with the reference point at the circle's center. In this case, objects are "grown" by the radius of the circle, effectively generating r-coverings of the objects, where r is the circle's radius.

Graph search of representations of free space. - As asserted previously, there are inherent limitations to finding paths in a discrete space. Both the recursive visibility search and the discrete space search methods are limited by their knowledge of the environment. Neither method performs well in environments containing complex objects [Fig. 49(a)]. By modeling the free space in an environment, some of these limitations can be avoided while improving the nature of the paths generated. The best example of this method is the generalized cone representation [see Brooks 1983a and Figure 49(b)]. In this approach, the environment description is preprocessed before any path is sought. The planner identifies the object sides that are relatively facing each other. The space between each such pair is known as a freeway through space. All of these freeways are found and analyzed to be sure that other obstacles do not intrude on them. The freeways are pared down according to their intersections with other objects in the environment. Those that will not accommodate the volume of space swept out by the moving object are discarded. The shape of the freeways is described as generalized cones. It is assumed that the moving object will be centered on the axis or "spine" of a cone as it moves along it. The freeways thus constitute the set of motions available to the moving object. The spines of the free space are used in planning a path for the object. As indicated in Figure 49(b), this approach generates paths that tend to keep the moving object as far from stationary objects as possible.



(a) Paths generated by discrete space (left) and visibility lines (right) approaches for a more complete object.



(b) Generalized cones generated by three objects and boundary.

Figure 49.- Path planning in complex environment.

Path-Planning Research Under the ROBSIM Contract

Three approaches to manipulator path planning were investigated under the ROBSIM contract:

1) The generalized cone method;

2) The joint space method;

3) The incremental, constrained-motion method.

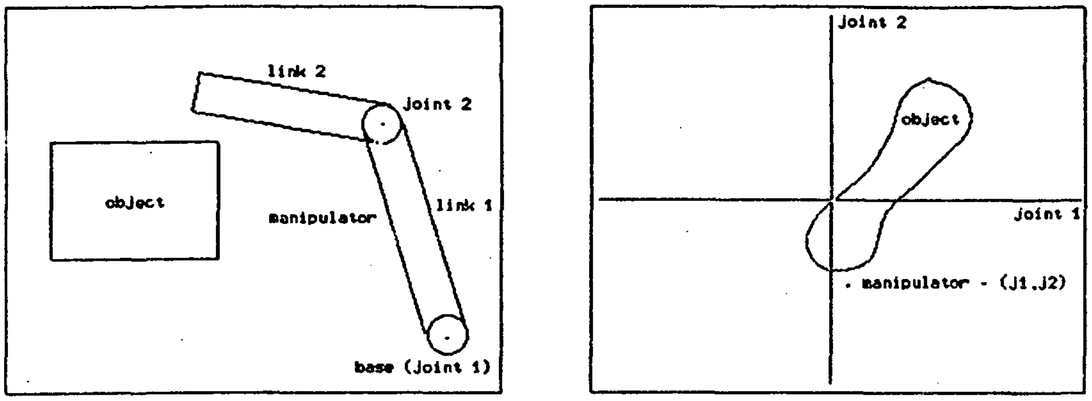
Research applied to these methods resulted in preliminary implementations of each approach. The following is a discussion of each method, its implementation, and the outcome of the research.

The generalized cone method. - A path planner based on the approach developed by Brooks (see [Brooks 1983a and 1983b]) and the previous discussion), in which free space is represented by generalized cones and trajectories along the spines of these freeways are generated, was implemented initially. To decrease the amount of computation involved in the algorithm, some simplifications are imposed on the planning environment. All objects in the environment, including the arm, are assumed to be rectangular in shape and grid-aligned (i.e., their sides are parallel to the sides of the workspace). This limitation is not overly restrictive. Irregularly shaped objects can be enclosed within bounding boxes. Few feasible solutions will be excluded by the use of this approximation, especially considering that this planning algorithm prefers paths that follow wide lanes rather than paths for which the arm barely fits between obstacles.

Programs for testing the algorithm were also developed. These programs generate random obstacles, start points and goal points within a two-dimensional workspace. Any number of objects are allowed in the environment. Four obstacles provide good testing results; with more obstacles, the computation time for preprocessing the environment grows rapidly. This problem is inherent in the Brooks algorithm but might be circumvented by applying heuristics to determine which calculations are necessary.

The planner was quite successful in solving the problems it set for itself. After generating a workspace, it would preprocess the environment, establishing the path freeways, and seek a path from the start toward the goal. Although it found a path in the majority of cases, the paths were sometimes very indirect. This may be caused by some subtle programming bug yet to be uncovered. The planner is able to rotate the arm as well as translate it, and occasionally finds intricate paths involving sequences of rotations and translations through several corners.

The joint space method. - Joint space is an N-dimensional vector space, where N is the number of manipulator joints or degrees of freedom, in which each coordinate axis corresponds to one joint displacement. Any point in this space represents a unique configuration of the manipulator. Similarly, with appropriate bounds on the joint displacements, any configuration of the arm corresponds to a unique vector in joint space. If, for any configuration of the manipulator, some link collides with an obstacle (including another link), that configuration is forbidden. In the joint space method of path planning, all obstacles and position constraints are mapped into forbidden regions in joint space. The problem of finding a collision-free path for the manipulator therefore reduces to that of finding a point-path (curve) through joint space that avoids forbidden regions (Fig. 50). Although the initial configuration (starting point) is uniquely defined, the target point is often not unique and might even constitute a goal surface.



(a) Physical space composed of manipulator and object.

(b) Corresponding angle space representation.

Figure 50.- Joint space method of path planning.

The attraction of this approach to path planning is that a relatively simple (and fast) search algorithm, such as the recursive visibility lines method outlined earlier, may be used to find paths for the entire manipulator. This method's main drawback is that the translation algorithm for mapping obstacles from physical space into their joint space representations can be exceedingly complex and computationally time-consuming. This can be attributed not only to the complexity of the inverse kinematic solution (i.e., finding joint displacements for a prescribed tool position), but also to the fact that collisions between the obstacle and all points of the manipulator (not just the endpoint) must be mapped. Therefore one point on an obstacle maps into one or more irregularly shaped regions in joint space.

Another limitation is that search routines such as the visibility lines method do not generalize directly to higher dimensions. This is because the routine generates paths from vertex to vertex in avoiding an obstacle. In two dimensions this can ensure an optimal path; in three-dimensional space it almost never does. This is because in higher dimensions, the shortest path around an obstacle usually involves crossing its surfaces at points other than vertices. Although methods that discretize three-dimensional edges have been used to overcome this problem (see [Lozano-Perez 1979]), these methods suffer from the same drawbacks as those discussed earlier.

Alternatively, the procedure can readily be modified to find optimal paths as described in the following discussion. Rather than using only vertices of the obstructing surface, consider instead the edges of the surface (Fig. 51). The points (C,C') along this edge that minimize the total distance traveled from A to the edge to B are readily evaluated as the solution of a set of linear equations. When more than one obstacle face is encountered along the path from A to B, successive application of this approach leads to a suboptimal solution. Alternatively, a modified quadratic programming algorithm can be employed to solve for an optimal solution. Hueristic or search methods can be applied to determine which obstacle edges to consider.

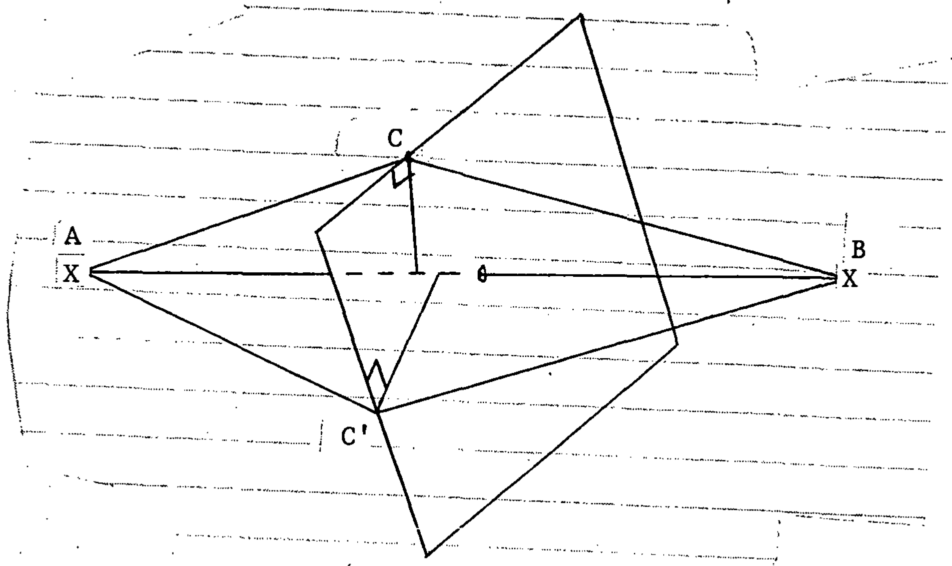
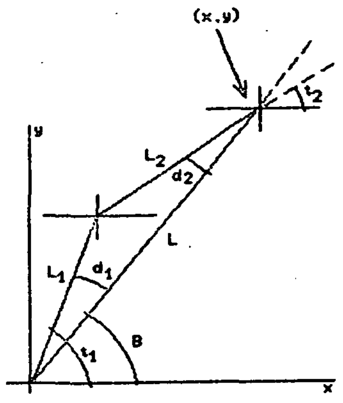


Figure 51.- Visibility line method in three dimensions.

Another method that holds promise for implementing the joint space method is based on a simplified representation of the obstacles. A more simplistic representation of the forbidden zones in high-order spaces could be the key to generating optimal paths. Representations that break up zones into regions of free or full rectangular parallelepipeds (in much the same way as the discrete space representation) facilitate the use of very simple interference detection algorithms, and could be used to find near-optimal paths in higher dimensional spaces [Haralick 1983].

Improved methods such as these are being investigated for extending the joint space method to higher dimensional spaces.

A preliminary implementation of the joint space method was developed for a manipulator of two joints. For a two-link manipulator with two rotational joints, there are two joint angle solutions for any given end-effector position in the reachable space. Figure 52 lists the translation functions that determine values for the joint angles corresponding to a given end-effector position. None, one, or possibly both of the joint angle solutions may be found to cause interference with an object. The complete mapping of an object results in a closed curve in the angle space. This curve is approximated initially by a large number of line segments. Next a minimal bounding box is generated enclosing all these segments (and thus the curve). Finally, an axis-oriented bounding box is generated to be used in rough positioning. Figure 53 shows the resulting hierarchical joint space representation.



$$L_1^2 = L_2^2 + L^2 - 2L_2L\cos(d_2) \qquad d_1 = \arccos[(L_2^2 + L^2 - L_1^2)/2L_2L]$$

$$L_2^2 = L_1^2 + L^2 - 2L_1L\cos(d_1) \qquad d_2 = \arccos[(L_1^2 + L^2 - L_2^2)/2L_1L]$$

$$t_1 = B + d_1 \quad t_1' = B - d_1 \qquad B = \arctan(y/x)$$

$$t_2 = B - d_2 \quad t_2' = B + d_2 \qquad L = \sqrt{x^2 + y^2}$$

t_1' and t_2' are the reflections of t_1 and t_2 about their respective axis of symmetry.

Figure 52.- Translation functions from physical to angle space.

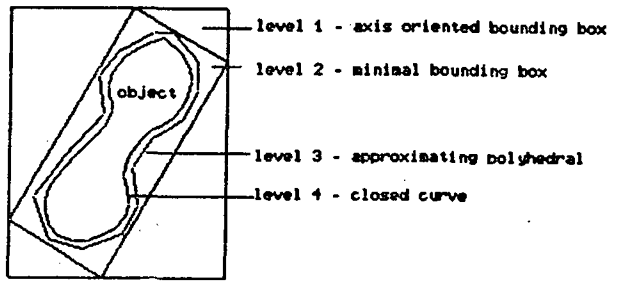
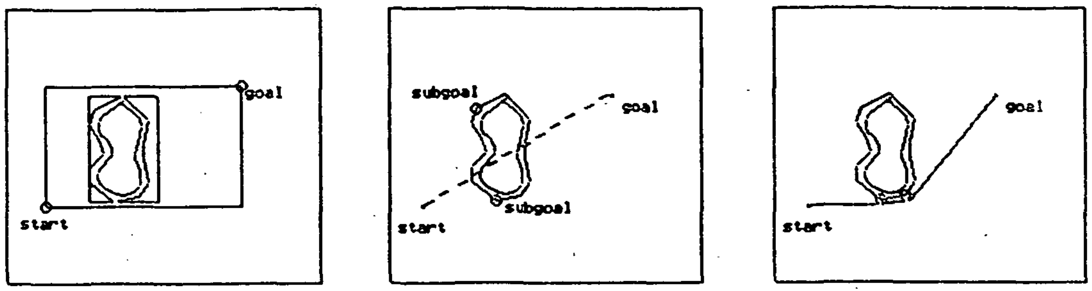

object
level 1 - axis oriented bounding box
level 2 - minimal bounding box
level 3 - approximating polyhedral
level 4 - closed curve

Figure 53.- An hierarchical object representation.

Once all objects in the physical space have been represented in joint space, the path-planning operation begins. The start and goal configurations of the manipulator are the start and goal points in joint space. The planner determines which "objects" (called "zones" in the joint space) are relevant by a simple bounding box intersection check [Fig. 54(a)]. Next, a straight-line path from start to goal is proposed. If a forbidden zone interferes with the proposed path, subgoal points are generated to allow the planner to avoid the offending zone. Thus the search operation of the planner follows the path of the recursive visibility lines method (Fig. 54).

It was thought that an indepth study of the simple two-joint case would yield insight into solutions in the general case. However, the generalizability of this method to higher degrees of freedom is inhibited in two important areas. First, the closed-form translations of objects in the physical world to their joint space representations are difficult to determine, especially for manipulators with several degrees of freedom. One method to overcome this problem involves the notion of incremental learning of the physical environment. This will be discussed in the final subsection.

The second drawback to the joint space method is that search routines for higher order spaces tend to generate nonoptimal solution paths. As described earlier, algorithms are being developed to overcome this limitation.



a) relavent objects found via rough check b) proposed path and generated subgoals c) the resulting path

Figure 54.- Path planning in joint space.

Initially, the joint space method for manipulator path planning seems attractive. Once a static space has been represented in the joint space for a given arm, the problem of finding a path for the arm is reduced to a much simpler and faster dynamic graph search problem. The problems inherent to this method provide some genuinely difficult research areas, both in geometric modeling and graph search techniques. However, for planning in a static environment for an arm with three degrees of freedom or less, this method is viable. This is particularly true for applications demanding that plans be generated in a short amount of time.

The incremental, constrained motion method. - This subsection describes a new approach to manipulator path planning in which the end-effector (or hand) moves toward the goal position in incremental steps. At each step, a check for potential collisions with obstacles is made. If any exist, the motion at each step in constrained to avoid the obstacles, while still moving toward its goal. A linear programming algorithm is used in this case to solve for the joint motion increments. The constraints and optimization criteria are transformed to joint space for each instantaneous position of the device. The two-dimensional case is implemented and used as a basis for discussion in the following paragraphs, although the method extends directly to three dimensions (nonplanar operations).

All obstacles are defined by enclosing line segments. The line segment representation is stored as a pair of endpoints and the outward-facing unit normal. Assume the manipulator is in an allowable initial or intermediate configuration. First, the world coordinate positions of all links and points of interest in the arm are evaluated as discussed in the Analysis Tools section of this report. Then, all potential collisions are identified. Figure 55 illustrates the three types of collisions possible in the planar case. These types involve (1) point on manipulator with edge of obstacle, (2) vertex of obstacle with link of manipulator, and (3) point on manipulator with vertex of obstacle.

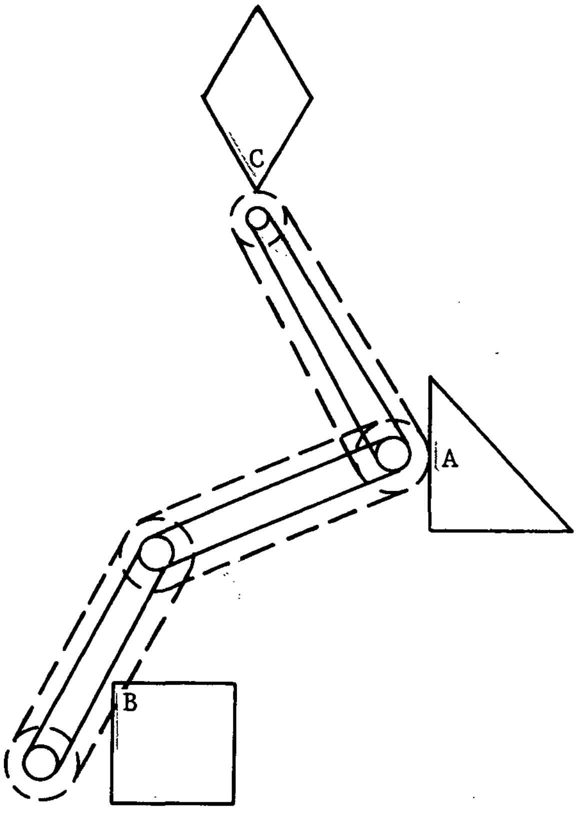


Figure 55.- Arm position in cluttered environment.

User-specified bounding covers (ϵ-covers) are defined around each manipulator link and point of interest. Potential collisions are identified by checking all manipulator points against all obstacle edges to see if they are within a distance E of each other.* Also, all obstacle vertices are checked for proximity to manipulator links. This checking process involves two steps: (1) evaluate the distance between the point and the line containing the edge segment; and (2) if this distance is less than E, check that the point is within the segment of this line defined by the edge endpoints. Potential collisions of type (3) are determined as an intermediate result of this check.

For each potential collision identified, a corresponding constraint is "activated" to prevent the collision from occurring on the subsequent motion step. Consider the point/line interaction shown in Figure 56.

* The efficiency can be improved by limiting the point-edge pairs checked to those that lie within the same discrete neighborhood.

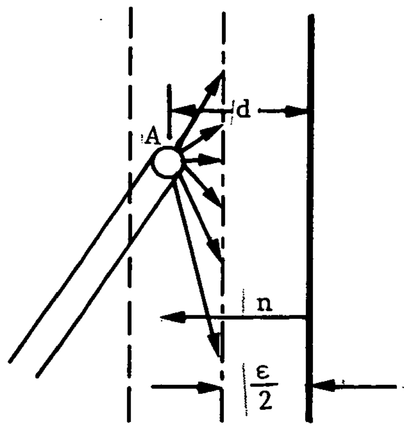


Figure 56. - Constrained motion of joint.

Because point A is within distance ε of the line, the constraint is activated. The constraint placed on the relative motion of A toward the line is that A cannot move closer than a distance ε/2 from the line. This method is employed to prevent A from jumping in and out of the ε-cover, repeatedly activating and deactivating the constraint. This constraint can be written

$$\Delta \underline{r}_A \cdot \underline{n} \geq \frac{\varepsilon}{2} - d$$

where $\underline{n}$ is the outward-facing unit normal, d is the current distance between the point and the line, and $\Delta \underline{r}_A$ is the incremental motion of point A relative to the line for the subsequent step. This constraint is transformed to joint space by the relations

$$\Delta \underline{r}_A = [J_A] \, \Delta \underline{\theta}$$

$$\underline{n}^T \, [J_A] \Delta \underline{\theta} \geq \frac{\varepsilon}{2} - d$$

where $\Delta \underline{\theta}$ is the vector of incremental joint displacements and $[J_A]$ is the translational Jacobian for point A, which is discussed further in the Analysis Tools section. Constraints of type (2) are handled identically, considering the moving link fixed and relative motion of the points of the environment. For constraints of type (3), the line normal $\underline{n}$ is replaced in these equations by a unit vector directed between the interacting points.

The incremental motion $\Delta \underline{\theta}$ is therefore constrained by a set of linear inequality constraints. Additionally, limits are placed on the magnitude of the change in each joints displacement

$$|\Delta \theta_i| \leq \Delta \theta_{i,\max}$$

To determine the joint motion step $\Delta\underline{\theta}$, an error vector $(\Delta\underline{r}_p^T, \Delta\underline{\phi}_p^T)^T$ between the current hand position and the desired hand position is computed as described in a previous section. Then, the equations

$$[J_p]\Delta\underline{\theta} = \left\{ \begin{matrix} \Delta\underline{r}_p \\ \Delta\underline{\phi}_p \end{matrix} \right\}$$

are solved, subject to the joint stepsize limits, using the linear equation solution algorithm previously discussed. This provides straight-line motion of the hand toward the goal. If, however, the resulting joint displacements violate any of the obstacle-avoidance constraints, the equations are solved by a different method.

In this case, all activated joint motion constraints are considered and the simplex algorithm for linear programming is applied. This is analogous to the method described for a similar manipulator path-planning problem [Grechanovsky 1981]. The linear optimization criterion is defined by

$$\text{maximize } \{\Delta\underline{r}_p \cdot \Delta\underline{r}_{p,\text{actual}} + \Delta\underline{\phi}_p \cdot \Delta\underline{\phi}_{p,\text{actual}}\}$$

which is translated into joint coordinates to give

$$\text{maximize } \{(\Delta\underline{r}_p^T, \Delta\underline{\phi}_p^T)\ [J_p]\ \Delta\underline{\theta}\}$$

The joint angles are updated using the resulting vector

$$\underline{\theta}_{new} = \underline{\theta}_{old} + \Delta\underline{\theta}$$

and the entire process is recursively repeated until the goal is reached.

Initial testing verified that this algorithm works very well as a low-level planner in an overall planning hierarchy. However, the manipulator can easily be trapped against the edge of one or more constraints, endlessly cycling among a few positions without making any progress toward the goal. This problem illustrates the need for a high-level planner that provides intermediate goals to the incremental-motion planner and monitors its operation.

The ROBSIM path planner contains an initial implementation of such a high-level planner. The Cartesian position space of the hand is discretized into a finite number of regions. Weights are assigned to transitions between contiguous regions; these weights indicate the relative ease of making the hand motion corresponding to that transition (small values indicate the transition is readily achieved). The high-level planner chooses a goal-directed path, defined as a sequence of regions to be traversed, based on these weights. The center point for the next region along the path is specified as the current goal for the low-level planner. The high-level planner then monitors the execution of the lower level.

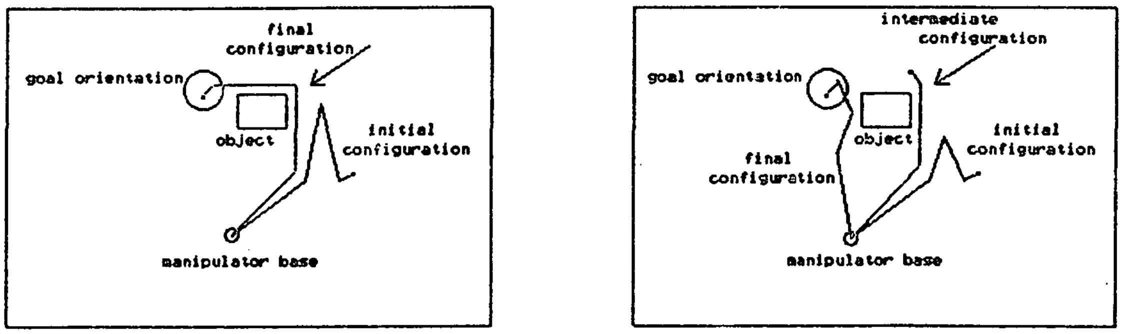
If the goal region is attained within a small number of steps at the lower level, the weight corresponding to that transition is decreased and the next target region is prescribed. If the goal region is not attained within a certain number of lower level steps, the weight for that transition is increased and the high-level path is resynthesized. In this manner, the high-level representation of obstacles is adaptively learned as the arm executes planning motions. This high-level planner prevents the low-level planner from becoming entrapped. This hierarchical design has provided good test results although the current graph search algorithm in the high-level planner needs modification to improve the overall planner performance.

Other high-level planning approaches in addition to this simplified discrete-space method could also prove productive. For example, a rule-based planner could be implemented.

Rules in the high level are simple heuristics pertaining to possible arm motions. For instance, a rule to determine a direction to be taken (by the manipulator's end-effector) in avoiding an object is

```
    (can-reach-around $arm $obstacle $goal-orientation) -
      (propose-path $arm $obstacle BEYOND)
```

This rule says that if the arm ($arm) can attain its final orientation ($goal-orientation) by reaching around the obstacle ($obstacle), then a sequence of subgoal configurations should be generated allowing the manipulator to avoid the object by moving BEYOND it. A situation in which this rule may be used is shown in Figure 57(a). Moving around an obstacle means that the overall length of the arm increases, rather than shrinking toward the base of the arm [see Figure 57(b)]. Another general heuristic says that shrinking the arm is preferred to extending it because a smaller arm tends to interfere with fewer objects than an extended one. Another rule points out that any object that interferes with link i may also interfere with links i+1, i+2,..., L (where L is the total number of links). The rules of the high-level planning system should be simple and their interactions limited to keep the planning speed of the overall system high.



a) manipulator reaches around object to attain goal

b) smaller manipulator cannot reach around; finds path beneath the object

Figure 57.- Actions of a rule-based planner.

The high-level planner does not attempt to find complete collision-free paths in any sense. Rather, it guides the operation of the low-level planner using information about the current state of the manipulator to generate plausible paths. This system may attempt ultimately impossible paths for some time before deciding on a new approach. However, when this happens, the system can be capable of large "jumps" back to previous manipulator configurations, particularly if the planning is completed before the motion begins. This backtracking scheme is maintained by the high-level system. Thus, when an attempt to reach a goal orientation becomes less desirable (or fails) in the current path direction than in some previous one, the system may return attention to the previous path, possibly jumping back over many attained configurations.

Comparison of Current Methods and Directions for Further Study

Three methods stand out as viable solutions to the problem of manipulator path planning--the generalized cone representation of free space, modeling physical environments in joint space, and incremental, constrained-motion planning. Each of these methods is directly applicable to manipulator path-planning problems. However, depending on the application, one method may be more suitable than another. In this section, each of these methods is discussed in terms of its strengths and weaknesses in various path-planning domains. Following this is a brief discussion of directions for further study.

The generalized cone representation of free space. - Recall that representing physical environments as a set of intersecting generalized cones involves finding cones between relatively facing edges (or faces, in three dimensions) of all the objects in the environment. The time required to generate the representation grows rapidly with the number of objects.* For this reason, the method is limited to applications involving only sparsely populated environments. However, if objects may be represented in terms of simple geometric figures (such as bounding boxes or rectangular parallelepipeds), evaluation of the free space becomes more efficient.

Another limitation of this approach is that solutions are determined solely through examination of possible freeway paths. This leads to the same problem as the discretizing of an environment; namely, the algorithm may not find a path even when one exists. This is because the object moving through the environment must remain aligned on a freeway at all times. Additionally, rotation is allowed only at freeway intersections (i.e., within the intersection of two cones). Thus, the planner may fail to find paths that require rotation between two intersection points. If the distance between objects in the environment is relatively large in comparison with the size of a manipulator's links, this is only a minor problem.

* According to Brooks, the algorithm has complexity (n3) "n cubed" in number of objects for 2D worlds; (n4) "n to the fourth" for 3D (see [Brooks 1983a], pps. 194, 196).

Modeling physical environments in joint space. - The most limiting aspect of this approach is calculation of the joint space representation of the physical world. Because there are many ways for a real manipulator to attain a given end-effector position, an a priori determination of all possible arm interferences is difficult, if not impossible. However, a preliminary determination involving the first three of a manipulator's joints (base, shoulder, and elbow) may be made, providing an approximation of free space suitable for rough arm positioning. This approach alone is viable in applications involving manipulators with three degrees of freedom or less, or those that do not require accurate arm positioning.

A combination joint space/linear programming approach would be viable in applications requiring accurate positioning of manipulators with more than three joints. Using this approach, the low-level linear optimization planner discussed would be coupled with a high-level planner that initially has no information regarding the physical environment. It would incrementally "learn" a joint space approximation of the environment as the low-level planner moved about and detected obstacles. This amounts to "feeling around" blindly in the physical world, with the advantage of remembering every position. In a cluttered environment, this system would initially find awkward paths. However, as the manipulator moved about more and more, the high-level planner would develop an accurate representation of the joint space that would be used to guide the arm along collision-free paths. This combination approach provides a means of modeling the joint space that avoids the complexity of a priori calculation. This adaptive, hierarchical path-planning algorithm could also accommodate changing environments.

The incremental, constrained-motion method. - This planning method is very promising, especially when coupled with an intelligent high-level planner that generates subgoals for the motion. No restriction as to the size of the environment or the number of components it may contain is imposed, nor is the system restricted to finding paths of a certain type. The number of degrees of freedom of arm motions is not limited and multiple arm systems can be accommodated. Sensor information can readily be incorporated in the description of the system and environment. Thus, this planner is especially well-suited for operation in situations where knowledge regarding the free space in an environment is either too costly to calculate or is impossible to determine ahead of time. This applies in either extremely cluttered static worlds or in dynamic environments.

Directions for further study. - Complex manipulator path planning in a static environment may be successfully accomplished using any of the three methods previously discussed. However, in any real world application of robotics technology, a manipulator will be expected to attain very accurate positions many times during a given task (e.g., inspection, assembly, or machine operation). Time constraints may also be imposed on the operation of the entire robotic system. These two characteristics require that a path planner generate plans in a very short amount of time. Some marriage between existing methods will be necessary to accomplish this. For example, the generalized cone method is capable of fast rough arm positioning in a simplified representation of an environment. The incremental, constrained-motion planner is capable of fine positioning in complex environments. Coupled with the adaptive learning of joint space, such a system could operate very well, especially in static environments. The system could switch back and forth between the two planning methods, using the generalized cone approach for large moves through uncluttered regions of the environment and the bilevel incremental planning system for smaller moves requiring finer positioning. The two operations could be done in parallel so by the time a plan to attain a rough configuration was completed, a path from there to the actual goal configuration would also have been generated. Such a planning system is being developed at Martin Marietta Denver Aerospace.

The main assumption of all present path-planning methods has been that the environment to be operated on does not change. For most real-world applications, this is an invalid assumption. Many applications, such as assembly, require the manipulator to move objects around in the environment. This is known as a dynamic application. The path planner may be constructed under the safe assumption that the system will be aware of all objects in its environment. The only constraint relaxed is that the obstacles remain stationary.*

This class of path-planning problems becomes even more complex when objects other than the manipulator may be in motion. For example, several manipulators may be working together in intersecting workspaces. The only technique suggested thus far for path planning in this situation is to allow only one arm to be in motion at a time whenever one is inside the working envelope of another. This amounts to assuming a static configuration for one (or more) arms while generating a path for another and forcing the former to assume that configuration and stop while the latter moves. One advantage of the incremental, constrained-motion method is that it can readily be extended to accommodate interacting arms, simultaneously controlling the manipulators to avoid each other.

One direction for path-planning research that would allow for less constrained operation in the case just described is toward execution of partial plans. This method is also viable in simpler applications with time-critical planning constraints. Several issues to be resolved include when to begin execution, physical "backtracking" of the manipulator when the current route is found to be impossible, and communication among path planners for the various moving bodies.

* There are two subclasses of this type of environment, one in which objects are moved only by the manipulator, and one in which objects may be moved by other means.

IMAGE PROCESSING AND VIDEO SIMULATION

A video simulation software module was developed to simulate the output of a video camera mounted on or near a simulated manipulator. A ROBSIM graphics representation depicting a scene of a PUMA 600 manipulator in a peg-in-the-hole scenario was generated with the ROBSIM program. Conversion of these data to a form used by the MOVIE.BYU package available on the Ramtek raster graphics device was performed. The single scene was then displayed using shaded graphics and color with the MOVIE.BYU package to generate a raster image of the scene; this image was then used as input to various edge detection algorithm modules. This method of ROBSIM geometry database conversion to MOVIE.BYU format may be used to generate a raster image of any ROBSIM scene on which new image processing algorithms, such as the edge detection and thresholding operations explained here, may be performed to enhance the resulting picture.

Edge Detection

An edge in a digital image is a change in intensities. However, an image contains many intensity changes, not all of which represent physical edges. For example, consider an image of a cube illuminated by a nearby point source. Intensity changes in the image occur at the intersection of two faces because of the change in the angle of reflection, and along each face because of the change in distance from the light source. In this example, a computer vision system must be able to distinguish the intensity changes caused by physical edges from those caused by other factors.

Edge detection schemes generally involve three stages of processing:

1) Edge pixels are detected;

2) Edge pixels are thinned to eliminate all but significant edges;

3) Edge points are linked to form lines, curves, etc.

Each of these stages is discussed in the following sections.

Detecting edge pixels. - Consider an image as a three-dimensional surface where each intensity value denotes the height of the surface at that point. Then changes in intensity represent changes in surface height, and the magnitude of an intensity change represents the magnitude of the surface slope. Hence, one approach to identifying the "edgeness" of a pixel is to treat the image as a surface and compute the magnitude of the surface slope at each pixel. On a continuous surface, this computation would involve computing the directional derivatives at each point. Because a digital image is a discrete function, difference operators are used to approximate the derivatives.

One of the first efforts at edge detection using difference operators was presented in [Roberts 1965]. Roberts computed an approximation to the magnitude of the surface gradient at each pixel by computing first differences (discrete first derivatives) in a 2x2 neighborhood of the pixel. Although the "Roberts" operator requires relatively few computations and produces sharp edges, it is sensitive to noise [Peli 1982]. Furthermore, the Roberts operator is directionally sensitive, having the greatest response to edges oriented at 45 and 135° and can only detect edges that lie to the right or below a given pixel.

The Roberts operator is an example of what is known as a 2x2 convolution operator. Let f(x,y) be the gray-level values at pixel (x,y) of an image. Then, the Roberts operator can be expressed as

$$R(x,y) = f(x,y) - f(x+1, y+1) + f(x+1, y) - f(x, y+1).$$

This is equivalent to convolving the image with the two templates

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \text{ and } \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

and taking the sum of the response magnitudes for the output at the pixel in the upper left corner. Thus the Roberts operator only takes differences to the right and below a given pixel and only along directions of 45 and 135° from the horizontal. Two more templates could be used to take differences at 0 and 90°. A larger operator must be used to take differences in a symmetrical neighborhood around a pixel. The smallest symmetrical operator is a 3x3 operator; such an operator can take differences at 0, 45, 90, and 135°. Increasing the size of the operators increases the number of directions along which differences can be taken. For example, a 5x5 operator can take differences at 0, 30, 60, 90, 120, and 150°. Because increasing the size of the operators also increases the number of computations that must be performed, a computer vision system designer must consider a tradeoff between directional sensitivity and computational cost for this kind of operator.

Instead of approximating the first derivative at a pixel by taking first differences, edges can also be detected by taking differences of average gray levels of adjacent neighborhoods [Rosenfeld 1982]. Such operators are less sensitive to noise than first difference operators but tend to produce wider, more blurred responses.

Second difference (discrete second derivative) operators can be used to detect edges. A commonly used second difference operator is the discrete Laplacian, which is relatively orientation-independent [Rosenfeld 1982]. Because difference operators respond to changes in the rate of change of intensities, in a region in which the image intensities vary smoothly, the discrete Laplacian gives no response. At the boundaries of smooth regions, however, the Laplacian responds twice--once on either side of the boundary, with responses of opposite sign. Edges are detected by locating all changes in sign, or zero crossings, of the response.

[Binford 1981] noted that a distinction should be made between the processes of detecting edges and localizing edges. A first difference operator performs well at detecting edges but responds over a broad region and is therefore poor for localizing edges. A second difference operator, however, performs well at the localization task but is more sensitive to noise [Rosenfeld 1982]. [MacVicar-Whelan 1981] reports edge localization to subpixel accuracy using a second difference operator.

[Prewitt 1970] first introduced a method for detecting edges by using the gradient of the best-fit polynomial surface in the neighborhood of a pixel. Using a least squares criterion, a polynomial surface is fit to a symmetric neighborhood around each pixel in an image. The magnitude of the gradient of the best-fit surface is used as the measure of the edgeness of each pixel. The size of the templates used is determined by the order of the polynomial surface to be fit. For example, a 3x3 Prewitt operator computes the gradient magnitude of the best-fit plane at each pixel, whereas a 5x5 operator uses the best-fit quadratic surface.

Edge detection can also be performed by fitting an ideal step edge to the neighborhood of a point. Generally, this technique is implemented using basis functions. The neighborhood of a pixel is expanded in terms of some orthogonal basis, and the coefficients of the expansion are compared with those for the expansion of an ideal step edge. By adjusting the coefficients of the ideal edge to minimize the squared differences between the sets of coefficients, the best-fit step edge through each pixel can be determined. [Hueckel 1971] was the first to use this technique by expanding regions in terms of the Fourier basis functions. [O'Gorman 1978] used Walsh functions. [Hummel 1977] used the Karhunen-Loeve expansion to derive a set of basis functions that are optimal for the detection of step edges. [Morgenthaler 1981] combined the techniques of surface fitting and step edge fitting to propose an edge detector using a local model of a step edge superimposed on a polynomial surface.

Thinning edge pixels. - The fact was pointed out earlier that one of the difficulties in edge detection is detecting only the significant edges out of all the edges present in an image. This section discusses some of the approaches that have been developed for thinning out the edges in an attempt to leave only the significant edges.

Decision-theoretic concepts can be used to thin edges [Rosenfeld 1982]). Suppose that a finite number of regions can be formed in an image, and suppose that the probabilities of a pixel being in each region or on a border are known. Then, if the probability densities of the edge detector responses in the regions or borders can be estimated, the probability of a given pixel being an edge can be computed.

The magnitude of the response of a first difference operator is proportional to the size of the intensity change at a point. If the assumption is made that large intensity changes are significant, one approach to thinning first difference operators is to threshold the edge picture at a particular edge strength, retaining only the edge pixels at which edge strength exceeds the threshold. However, this thinning technique fails if the strength of significant edges varies greatly over an image. For example, consider an image of objects in partial shadow. Object edges in shadow are faint and are likely to be lost if a global threshold is used. [Rosenfeld 1971] suggested suppressing all edges except the strongest for some distance taken perpendicular to the edge. [Hanson 1980] used local thresholding based on the distribution of edge strengths in square local neighborhoods.

[Rosenfeld 1971] also proposed a multistep technique for determining the "best" edge at a point. The technique requires taking the differences of averages using neighborhoods of many different sizes. Each of the resulting edge pictures is thinned by suppressing nonmaxima across the edge. Finally, a composite edge picture is obtained by comparing the different edge strengths obtained at each pixel. Let $e^{(h)}(i,j)$ be the edge strength (after initial thinning) at pixel (i,j) using a difference of averages operator of size hxh. The final edge strength at (i,j) is given by $e^{(m)}(i,j)$ where m is the largest value such that

$$\ldots e^{(m+k)}(i,j) < e^{(m+k-1)}(i,j) < \ldots < e^{(m)}(i,j)$$

but

$$e^{(m)}(i,j) \not< e^{(m-1)}(i,j)$$

The symbol "<" is taken to mean "much smaller than," and in the initial implementation of Rosenfeld and Thurston, meant roughly "less than three-quarters of." This algorithm is capable of detecting texture edges and was shown in [Peli 1982] to be fairly tolerant of noise.

Smoothing an image before applying an edge detector can reduce the effects of noise. Many kinds of smoothing operators exist and are discussed in texts on image processing (see, for example, [Rosenfeld 1982]). Operators can be designed that combine the operations of smoothing and edge detection.

[Binford 1981] proposed that a particular technique be implemented to smooth a picture for edge detection. He calls the technique "lateral inhibition." Implementation consists of subtracting from each pixel value the average intensity of the eight pixels immediately surrounding it. The lateral inhibition operation effectively suppresses the effects of smooth variations in intensity.

[Shanmugan 1979] proposed a frequency domain filter for edge detection. The proposed filter yields a maximum response in the vicinity of an edge. For the special case of a step edge, the optimal frequency domain filter is the second derivative of a Gaussian. Marr and Hildreth [Marr 1980] proposed an edge detector using the convolution of a Gaussian smoothing function with a Laplacian. The Marr-Hildreth edge detector employs operators of several different sizes to detect edges at various resolutions [Hildreth 1982].

The Marr-Hildreth edge detector produces good results but is computationally expensive; the templates used are huge by conventional standards--the smallest is 32x32 pixels [Brady 1982]. However, improvements in computer hardware technology may make even such large operators feasible for use in real-time applications. Nishihara and Larson designed a special-purpose processor for performing convolutions [Nishihara 1981]. The processor is capable of convolving a 1000x1000 image with the smallest Marr-Hildreth operator in about 1 second [Hildreth 1982]. Even greater speed can be expected as the state of the art in hardware improves.

Linking edge pixels. - After detection and thinning of edges have been performed, the next step is to link edges together to impose some order on the image. The ways in which edge pixels are linked depend on the kinds of linear features expected to be in an image.

Rosenfeld and Thurston [Rosenfeld 1971] proposed algorithms for detecting thin curves and wide streaks. Using their formulation, a point lies on a thin curve if it satisfies two conditions:

> "Condition 1: It has a pair of lower valued neighbors on opposite sides of it (in the direction across the curve).
> Condition 2: It has two other neighbors (in the direction along the curve that satisfy Condition 1."

Variations of this algorithm can be used to link pixels belonging to a variety of linear features.

[Shirai 1978] used gradient magnitude and direction, as well as a classification of edge type to link edges. The edge types used are shown in Figure 58. Linking proceeds in two stages. First, edge "kernels" or sets of pixels of the same type and direction, are identified. Then, each kernel is extended by tracking as far as possible in both directions. Tracking consists of predicting the position and gradient values of the next point, and adding pixels to the edge if their values differ from those predicted within prespecified tolerances. Predictions are updated every time a pixel is added, enabling the techniques to track curved edges as well as straight ones.

Ballard and Brown [Ballard 1982] advocated the use of the Hough transform to detect arbitrary curves. As an explanation of the technique, consider the example of detecting straight lines (following the development of Ballard and Brown). Let (x,y) be an edge pixel in an image. Any line through (x,y) is given by the equation $y = mx + c$. Because the set of all possible lines through (x,y) is represented by a line in m,c-space, each edge pixel in an image can be associated by a line in m,c-space; the set of edge pixels then corresponds to a set of lines in m,c-space. If two edge pixels lie on the same line in an image, their corresponding lines will intersect in m,c-space. Therefore, a straight line in an image will be represented as the intersection of many lines in m,c-space. By detecting clusters in m,c-space, lines in an image can be detected. The Hough transform technique can be extended to detect any kind of parameterized feature.

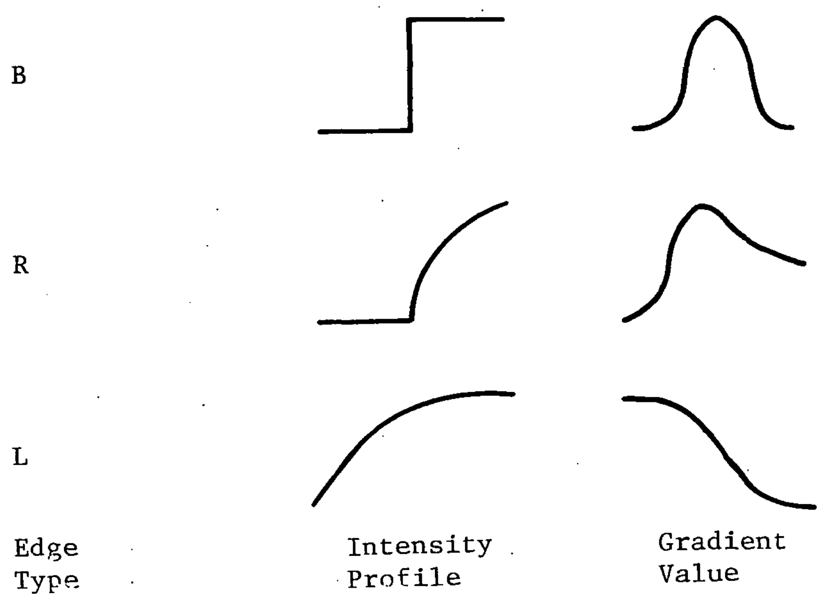


Figure 58. - Classification of edge types (after [Shirai 1978]).

Because edge detectors cannot be expected to give perfect results, gaps can be expected in linear features. The Hough transform can be used to detect features and fill in missing pixels. Another technique is to track features and fill in pixels that comprise gaps of a certain maximum size [Rosenfeld 1982].

59

The following pictures of the PUMA manipulator [Fig. 59(a) thru (h)] were taken from a Ramtek graphics terminal after conversion of an Evans and Sutherland line graphics display to a solid figure display.

"Page missing from available version"

pgs 135 and 136

SIMULATION VALIDATION

This section describes validation of the simulation developed under this contract. Two hardware systems were used to validate the ROBSIM software--a Martin Marietta two-degree-of-freedom planar arm and a six-axis PUMA robot. The sequence of validation tasks was:

1) Identify mass and motor parameters for the planar arm;

2) Model the planar arm in ROBSIM;

3) Employ the same voltages used to drive the planar arm motors to drive the ROBSIM response simulation and compare the planar arm positions with the simulated position as functions of time;

4) Repeat steps 1) through 3) for the PUMA robot.

Planar Arm Parameter Identification

The first step in simulating existing hardware was to determine the physical parameters that identify the system and its motion. Mass properties of the planar arm were first determined by taking the arm apart to find the lengths and weights of the arm components. Link centers of mass and inertias were then calculated from these measurements. Figure 60 shows the configuration of the planar arm used. It included a base, shoulder motor and gears, shoulder link, elbow motor and gears, elbow link, and bracket.

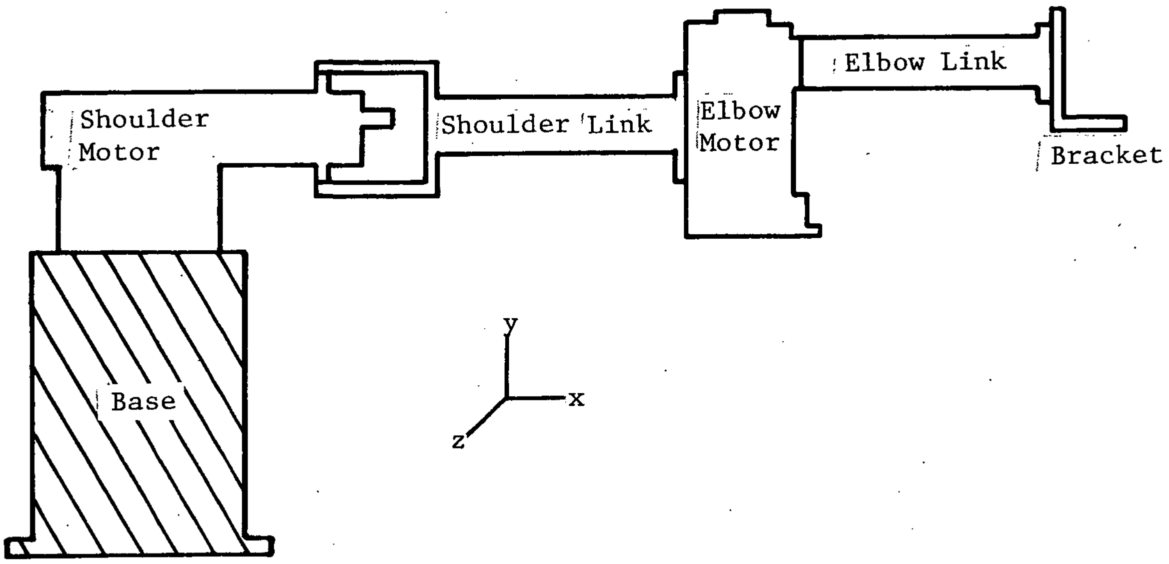


Figure 60. - Planar arm configuration.

Table V gives the values obtained for link masses, centers of mass, and y-axis inertias about the link's center of mass.

TABLE V. - PLANAR ARM LINK MASS PROPERTIES

	Mass, kg	Center of mass, m	I_{22} inertia, kg-m^2.
Shoulder	1.0297	0.1605	0.0132
Elbow (including bracket)	0.888	0.2095	0.0106

Motor parameters were obtained primarily from manufacturer's data sheets. However, the parameters that could be were tested. These included the torque constant, motor resistance, back-EMF, and static, coulomb, and viscous frictions. The torque constant test measured the force at a known moment arm while monitoring the motor current and voltage. Taking into account the amplifier gain allowed the torque constant to be calculated. Motor resistance was determined from the slope of the voltage vs current curve obtained in the torque-constant test. Back-EMF is the open-circuit voltage produced when driving the motor manually through the gearing. The shaft velocity was monitored and the EMF constant was found from the slope of the voltage vs velocity curve. Static friction, which is the torque necessary to begin joint rotation, was measured by increasing the voltage slowly up to the point at which joint rotation begins and noting this voltage. Coulomb friction, which is the torque necessary to keep the motor rotating at a slow speed, was determined by noting the lowest voltage that would keep the motor turning when given a small initial velocity. The actual static and coulomb friction torques were calculated from the measured voltages, amplifier gains and motor torque constants. Viscous friction, which is proportional to the motor angular velocity, was determined by giving the arm an initial velocity and then recording the angular velocity and displacement with an open circuit at the motor leads. Curve fitting these empirical data led to extraction of the coulomb and viscous frictions.

Before the validation was conducted, a decision was made to go to current control of the planar arm motors instead of voltage control. For purposes of the simulation, this takes the motor resistance, back-EMF, and inductance out of the control loop and they are set to 1, 0, and 0 respectively. Table VI lists the motor parameters used in the ROBSIM simulation at the beginning of the validation runs.

TABLE VI. - MOTOR PARAMETERS INITIALLY USED IN SIMULATION

	Shoulder	Elbow
Motor torque constant, Nm/amp	3.58	28.5
Motor gear ratio	20.25	86.4
Amplifier gain, amps/volts	0.5202	0.2543
Back-EMF constant, V/rad/s	0	0
Motor effective inertia, kg-m^2	0.0893	0.6763
Motor resistance, ohms	1	1
Motor inductance, henries	0	0
Coulomb friction coefficient, N-m	0.439	1.36
Static friction coefficient, N-m	0.616	2.04
Viscous damping coefficient, N-m/rad/s	0.0878	1.135

Simulation Comparison - Elbow

The comparison between hardware and software first looked at each joint separately and then at combined motion of both the shoulder and the elbow. The planar arm runs used were:

1) Elbow sinusoidal motion, $\pm 20°$ amplitude, 0.15-Hz frequency, 0.03-s sampling time;

2) Elbow trapezoidal motion, $\pm 20°$ amplitude, 0.15-Hz frequency, 0.03-s sampling time;

3) Shoulder sinusoidal motion, $\pm 30°$ amplitude, 0.15-Hz frequency, 0.03-s sampling time;

4) Combined shoulder and elbow motion, $\pm 30°$ amplitude, 0.1-Hz frequency, 0.03-s sampling time.

The simulation modeled the planar arm using the parameters in Tables V and VI. Response simulation runs were executed in ROBSIM using the same control voltage used to control the planar arm. To compare the motion of the simulation with that of the actual hardware arm, one of the ROBSIM postprocessing options was used. This option plays back the motion that occurred during an analysis run, and also simultaneously displays a stick figure whose motions represent the motion of the hardware arm.

The first hardware motion to be simulated was the sinusoidal elbow motion. Although the first attempt showed agreement in the direction of motion, the simulation amplitude and accelerations were too large. To attempt to obtain better agreement, the elbow motor torque constant and friction coefficients were reevaluated. The torque constant was determined by attaching the force/torque wrist to the arm and then increasing the voltage to the elbow motor and measuring the force exerted against a constraint. The slope of the force voltage curve multiplied by the amplifier gain and moment arm defined the motor torque constant. This testing obtained a torque constant of 12.8 Nm/amp for the elbow motor. The static friction torque was obtained by increasing the voltage to the elbow joint until the joint moved. This voltage was then multiplied by the amplifier gain and motor torque constant. The coulomb friction torque was evaluated by changing the voltage to find the lowest voltage that would keep the joint moving after it was given a small initial velocity. Multiplying by the amplifier gain and torque constant gave the actual friction torque value. This static friction torque was also evaluated by looking at the planar arm motor voltage and position plots. The voltage at which motion begins (at extreme reach of each cycle) was used to calculate the static friction torque. The coulomb friction torque is approximately two-thirds of the static torque. Testing the arm gave values of 1.5 Nm and 1.0 Nm for static and coulomb friction respectively. The voltage taken from the plot indicated static friction to be about 2.4 Nm. In testing both sets of values in the simulation, it was noted that the higher friction values seemed to give better agreement with the hardware positions. At this point the agreement between the hardware and the simulation was quite good. Both reached the same amplitude and would start motion in the new direction at the same time. However, the simulation would move out more quickly and reach the end of the motion and stop before the hardware would. This seemed to indicate that the effective inertia of the joint/link pair was too low. Because it was assumed that the link inertia was accurately known, it was assumed that the motor effective inertia was in error. The elbow motor inertia was increased until the motion of the hardware and simulation matched well. This meant increasing the inertia from 0.68 to 1.8. The positions of the elbow of the planar arm and its simulation are shown in Figure 61. (In the figure h denotes the hardware motion and s the simulation.) The negative shift of the simulation was because the arm was probably not exactly in the horizontal plane so gravity caused the motor drive voltage to be slightly greater in one direction than the other.

After the sinusoidal motion was matched, the voltages used for trapezoidal motion of the planar arm elbow were used to control a response simulation run of ROBSIM. Figure 62 shows the hardware and simulation elbow positions for trapezoidal motion.

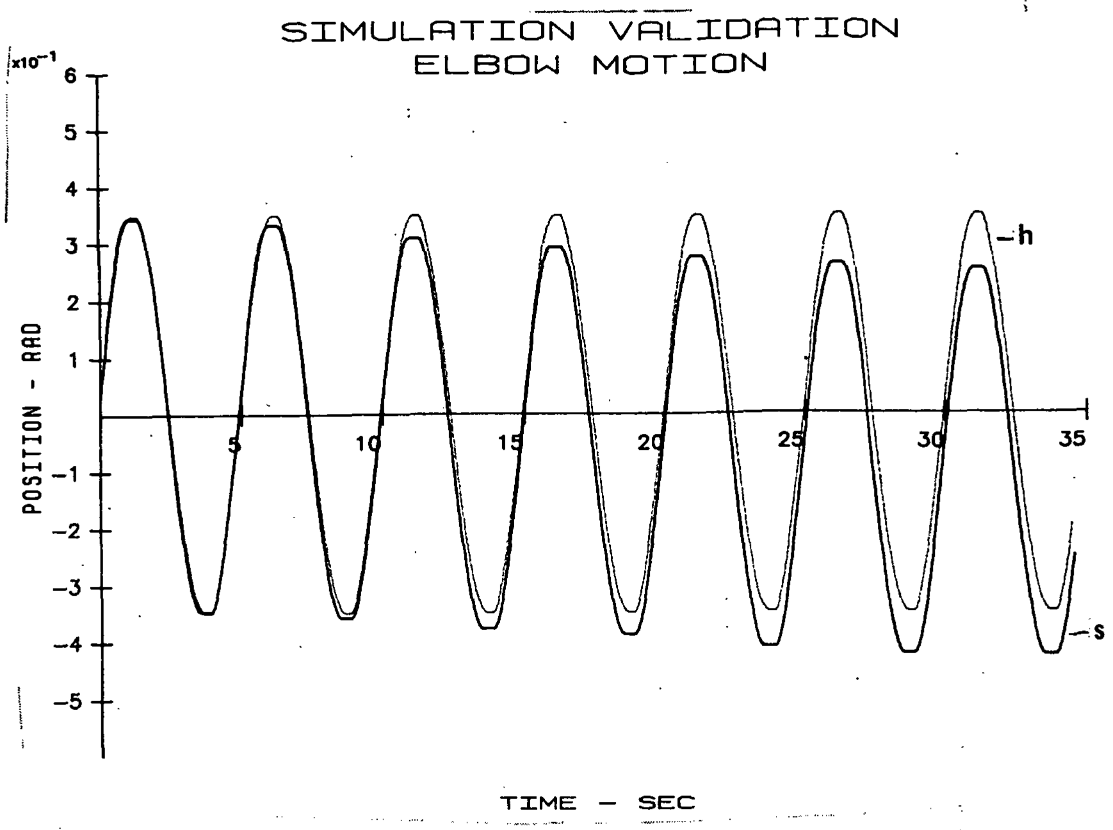


Figure 61. - Elbow simulation and hardware positions, sinusoidal motion.

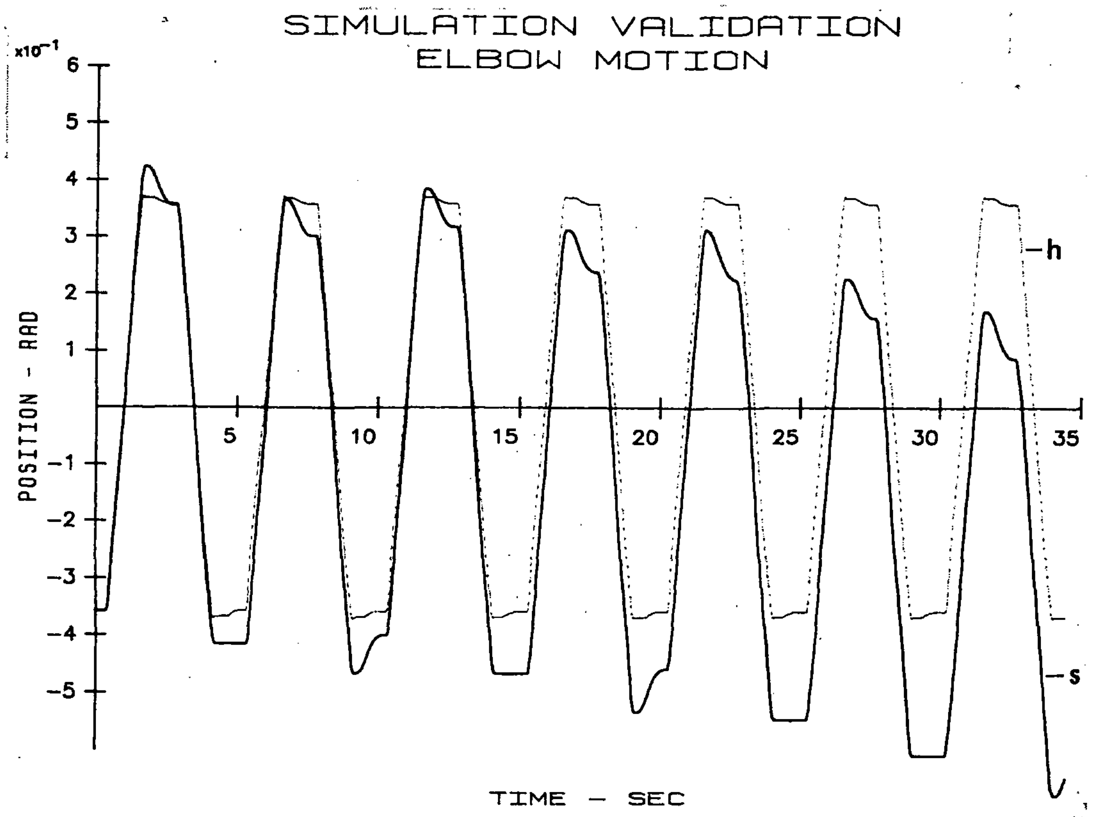


Figure 62. - Elbow simulation and hardware positions, trapezoidal motion.

The planar arm shoulder run commanded the joint to move 30° in both positive and negative directions. As with the elbow, simulation of the shoulder motion initially showed only minimal agreement with the actual hardware motion. The motor torque constant and the static and coulomb frictions were evaluated in the same manner as described for the elbow. Testing the planar arm gave a value of 2.6 Nm/amp for the shoulder motor torque constant and values of 0.95 Nm and approximately 0.88 Nm for the static and coulomb frictions respectively. The preceding friction torque values are for motion in the positive direction. Testing of the planar arm showed that frictions in the positive direction were much greater than in the negative direction. This was also evident by looking at the plot of shoulder motor voltages as a function of time. To compensate for the effects of the direction dependency of the friction torques, voltages corresponding to negative joint velocities were modified by subtracting an offset voltage. This value was determined by calculating the net positive offset per time step from the voltage data and subtracting twice this value. The 0.2 volt subtracted seemed to overcompensate for the problem. Trial and error seemed to indicate that approximately -0.18 volt was the best offset. In addition to offsetting the negative velocity voltages, the beginning of the voltage file up to the first zero velocity point was cut off to prevent any initial conditions from affecting the simulation run. After the modifications mentioned (motor torque constant, static and coulomb friction torques, and motor voltages), reasonable agreement was obtained between the planar arm shoulder motion and the simulation. But, as with the elbow, the simulation motion seemed to indicate a low effective inertia for the shoulder joint/link pair. Again it was assumed that the link inertia was fairly accurate and that the joint effective inertia needed to be modified. An inertia value of 0.45 kg-m^2 and an updated coulomb friction torque of 0.77 Nm produced good agreement between the simulation and the hardware it was modeling. Figure 63 shows both the hardware and simulation positions as function of time for the shoulder.

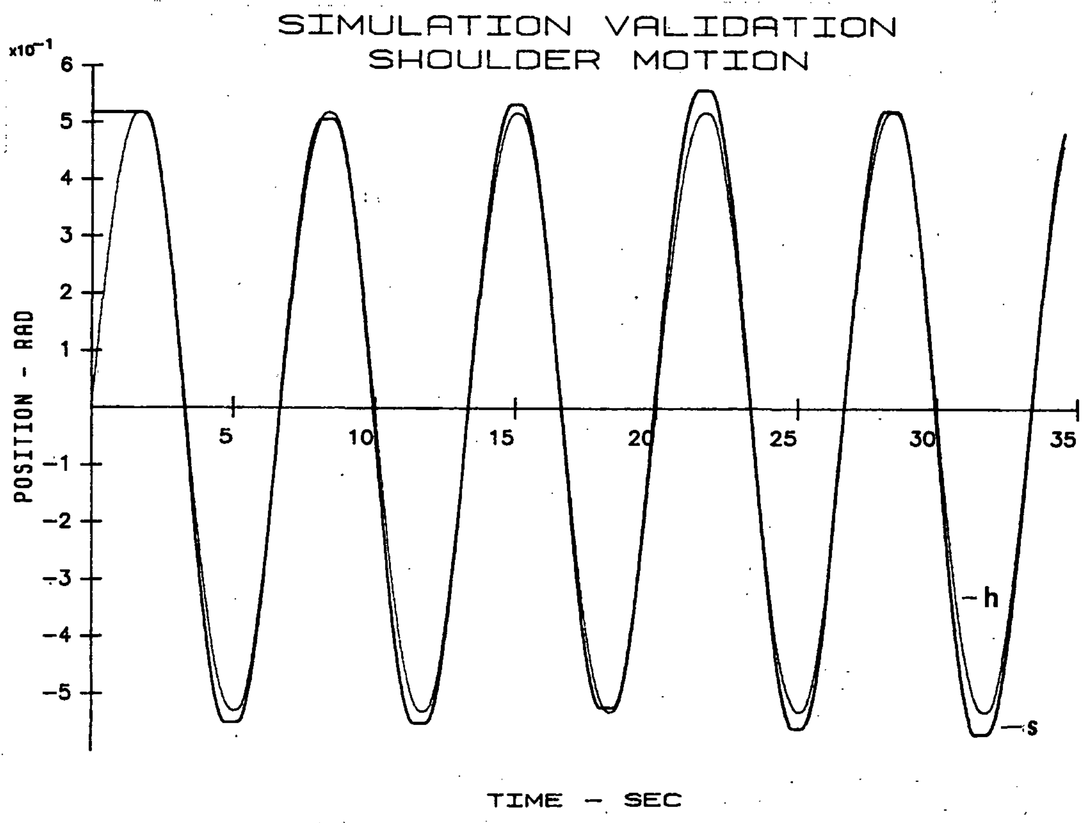


Figure 63. - Shoulder simulation and hardware positions.

Table VII lists the motor parameters for the planar arm at this point.

TABLE VII. - PLANAR ARM MOTOR PARAMETERS.

	Shoulder	Elbow
Motor torque constant, Nm/amp	2.6	12.8
Motor gear ratio	20.25	86.4
Amplifier gain	0.5202	0.2543
Back-EMF constant, V/rad/s	0	0
Motor effective inertia, kg-m^2	0.45	1.8
Motor resistance, ohms	1.0	1.0
Motor inductance, henries	0	0
Coulomb friction coefficient, N-m	0.77	1.5
Static friction coefficient, N-m	0.95	2.4
Viscous damping coefficient, N-m/rad/s	0.0878	1.135

Simulation Comparison - Combined Motion

Thus far, good agreement between the hardware and simulation has been shown for each joint (shoulder and elbow) separately. To further check the validity of the simulation, a planar arm test case that combined the motion of both the shoulder and the elbow was run. Both joints were commanded to move 30° in both the positive and negative directions at a 0.1-Hz frequency. The simulation was run using the motor and link parameters set previously when each joint was looked at separately. Figures 64 and 65 show the shoulder and elbow hardware and simulation positions.

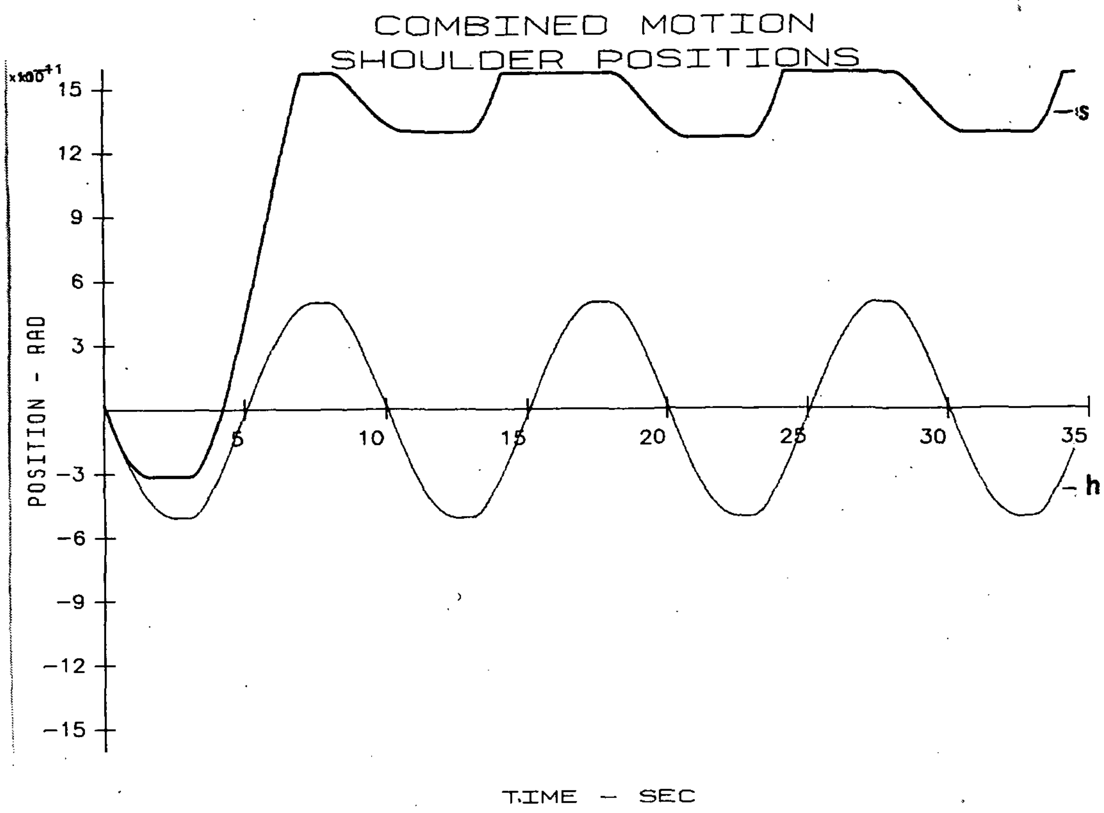


Figure 64. - Hardware and simulation shoulder positions for combined motion.

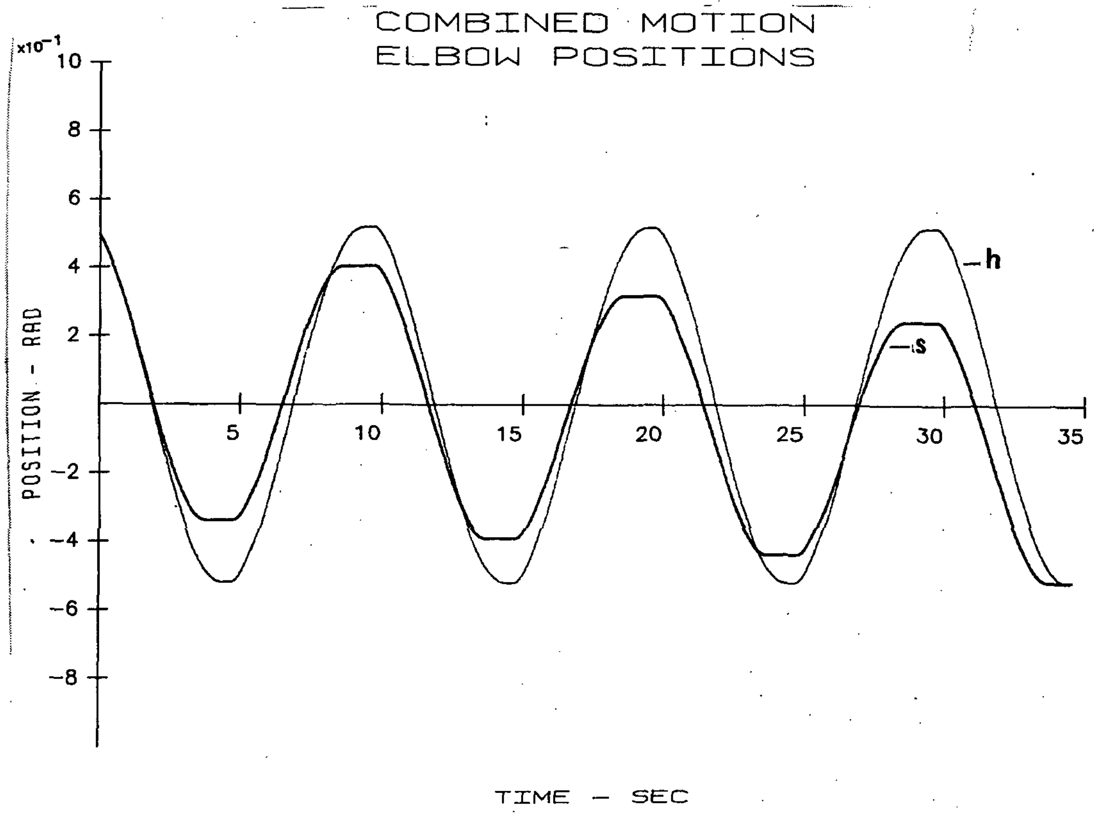


Figure 65. - Hardware and simulation elbow positions for combined motion.

As can be seen in Figure 64, the simulated shoulder position was forced to the positive limit because of the direction dependency of the shoulder frictions. The shoulder motor voltages corresponding to negative joint velocities were then offset by -0.18 (this value was determined earlier). The plot of shoulder hardware and software positions after offsetting the drive voltages is shown in Figure 66.

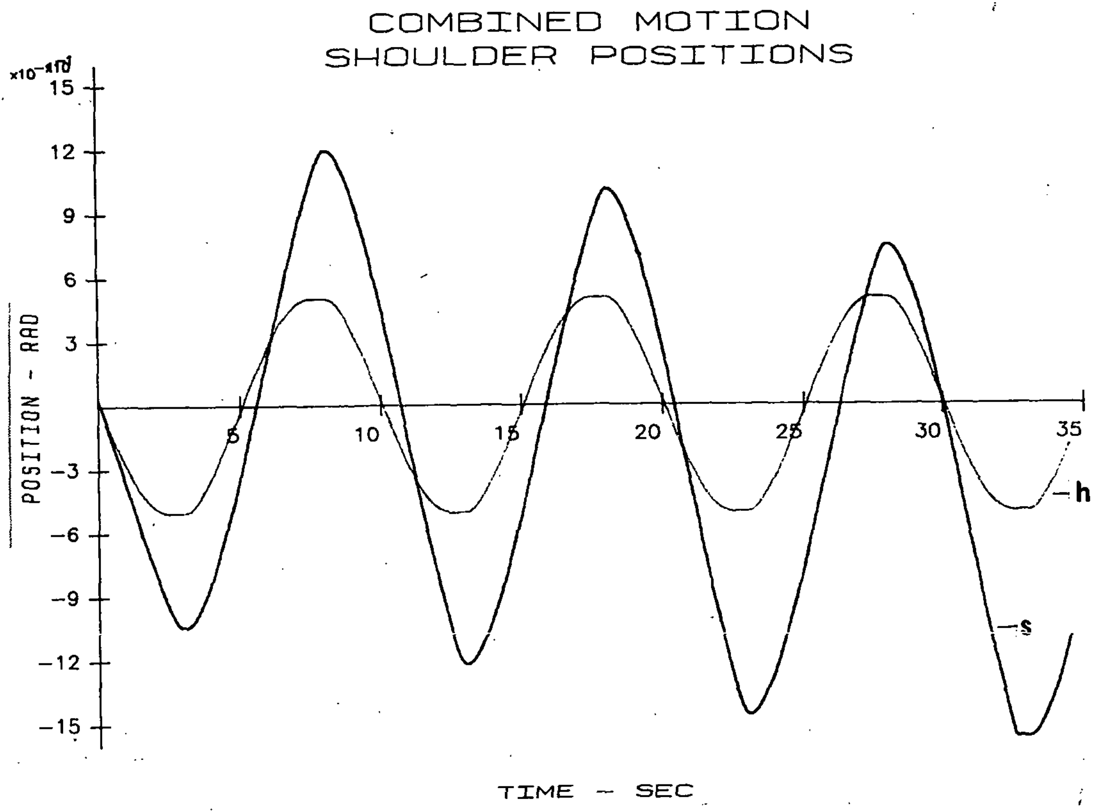


Figure 66. = Shoulder hardware and simulation positions after offset.

Figure 66 indicates that the effective inertia is too low and that the elbow link inertia must be increased. (Increasing the shoulder joint or link inertia would change its response when run alone.) To keep the motion of the elbow the same when run by itself, the elbow motor inertia would have to be decreased to keep the effective inertia of the elbow joint/link pair the same. As a first attempt, the elbow motor effective inertia was decreased to 1.0 kg-m^2 and the link inertia (about its center of mass) was increased to 0.81 kg-m^2. Although these values do not give the same total effective inertia as previously noted, they were used to see if this was a valid method for correcting the simulation model. Figures 67 and 68 show that this is the correct method for updating the simulation model, and that trial and error testing would yield an accurate model.

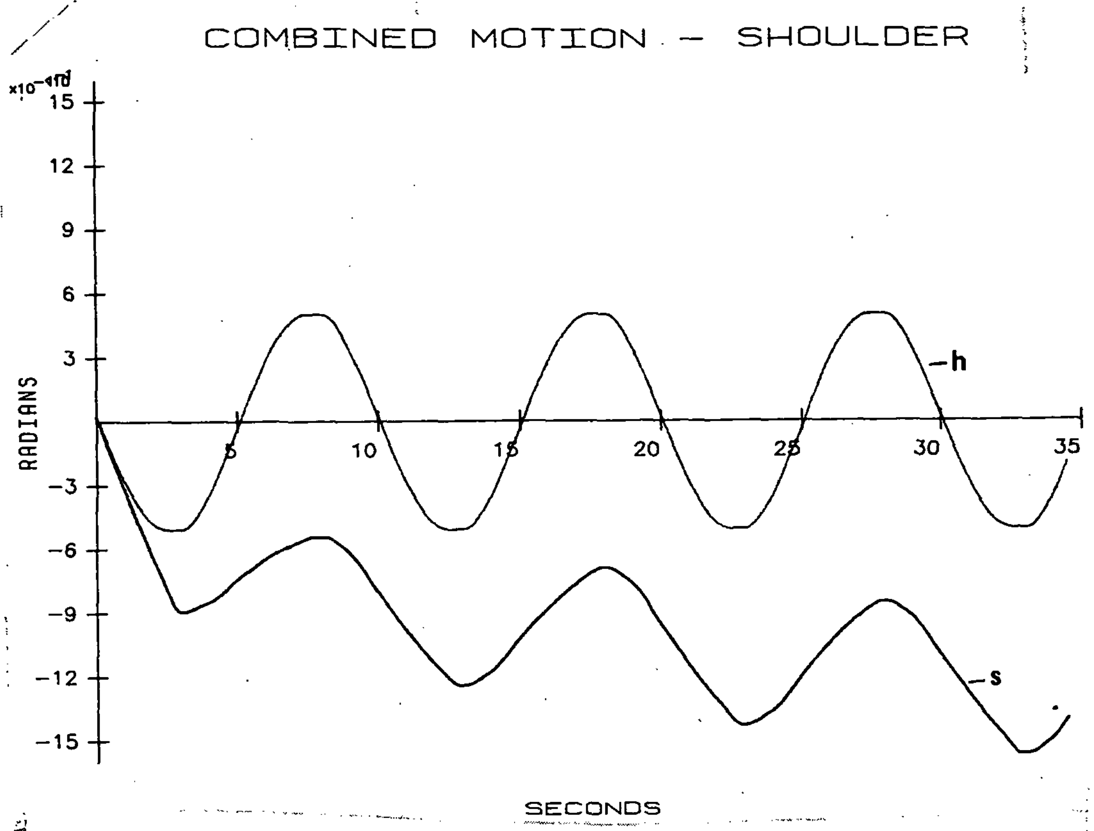


Figure 67. - Shoulder hardware and simulation positions, combined motion.

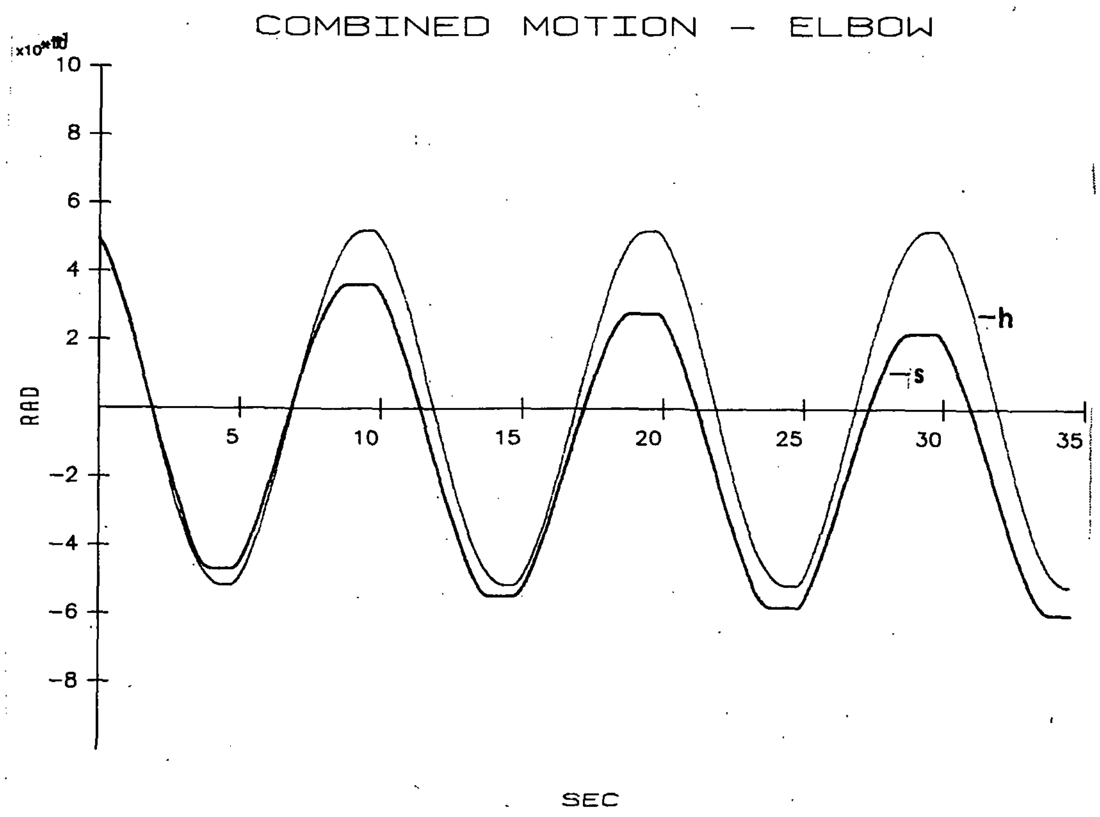


Figure 68. - Elbow hardware and simulation positions, combined motion.

Initial experiments were performed to determine the kinematic and dynamic parameters for one of the two PUMA robots at NASA Langley Research Center. [Paul 1983] presented kinematic and dynamic parameter estimates for the PUMA six-degree-of-freedom manipulator. [Lee 1984] determined the kinematic parameters of this arm more precisely. The evaluation described here was performed to validate these earlier results and to verify the consistency of parameters between devices of the same model.

Figure 69 illustrates the PUMA robot and the kinematic parameters to be identified. The relative angles between consecutive joint axes were all assumed to be 0° or 90° and no attempt was made to verify these values. To determine the length parameters, a point on each joint axis had to be located first. This task was performed by marking the position (in the world coordinates) of one point on a subsequent link. The joint was then rotated by 180° and the new position of the point determined. (Protractors and plumb bobs were mounted on some of the joints to verify that 180° rotations were achieved.) The midway point between these two positions locates a point on the joint axis (Fig. 70). Using this procedure, the kinematic parameters listed in Table VIII were obtained (see also [Paul 1983]. Table IX shows that these agree well with the valued obtained by [Lee 1984] and [Paul 1983].

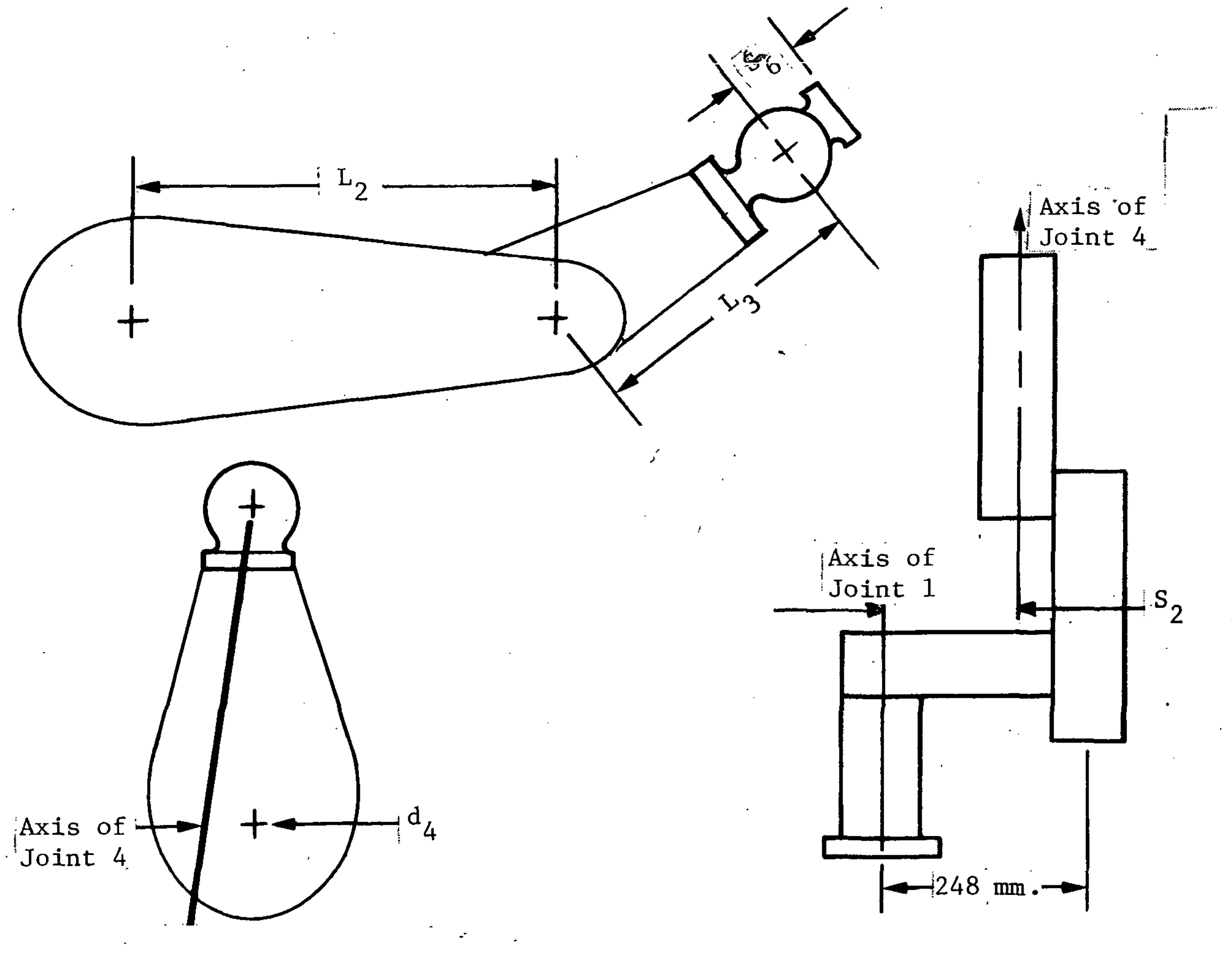


Figure 69. - Kinematic parameters of the PUMA.

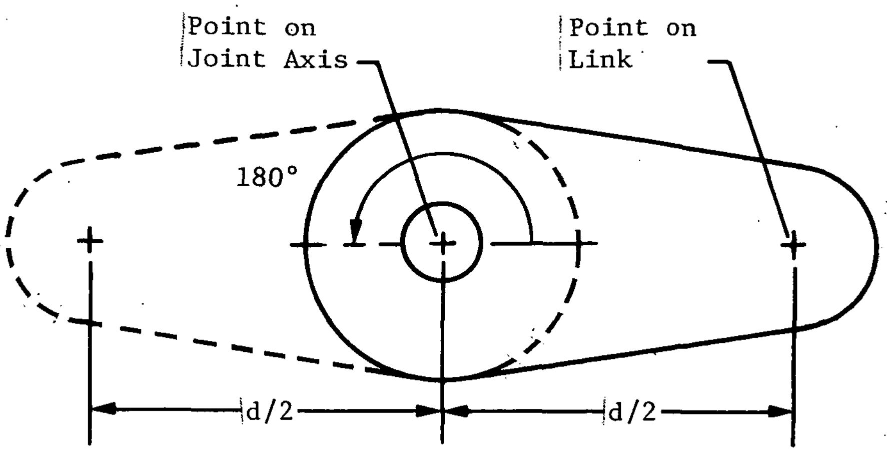


Figure 70. - Locating a point on the joint axis.

TABLE VIII. - PUMA KINEMATICS

Joint	α, deg	S	a, mm
1	-90	0	0
2	0	S_2 = 150.4	L_2 = 431.0
3	90	0	d_4 = 20.2
4	-90	L_3 = 433.1	0
5	90	0	0
6	0-	S_6 = 56.1	0

TABLE IX. - COMPARISON OF PUMA CHARACTERISTICS

Parameter	Measured Value*	[Lee 1984]	[Paul 1983]
S_2	150.4	149.09	125
L_2	431.0	431.8	432
L_3	433.1	433.07	432
d_4	20.2	20.32	19
S_6	56.1	5625	-

* All parameters in millimeters.

Based on the measurement technique, the accuracy of these values is estimated to be on the order of ± 0.5mm. Furthermore, many of the kinematic parameters have assumed values. Verifying these parameters requires a more elaborate experimentation and data reduction scheme such as that proposed in [Barker 1983]. We believe that high-accuracy positioning of robotic devices will require some type of end-effector position sensing or online calibration because of these kinematic uncertainties (as well as the fact that structural deformations occur under load.)

A computer simulation of the system response requires several associated dynamic parameters to be evaluated for the system. The dynamic parameters needed for the simulation include masses, inertias, friction forces, etc. They can be broken down into two general classes:

1) Parameters that can be identified by static force measurements;

2) Those that require dynamic response measurements.

First, static force measurements were performed to determine the link masses, link centroid locations, and friction torques at each joint. The measurements were performed by setting the robot up in several configurations, freeing one joint while leaving the relative motions at the other joints fixed and measuring the forces needed to start the arm in motion. For simplicity it was assumed that the centroid of each link lies at an unknown distance a_i along the link axis. The main concern was to identify the mass m_i and centroid location a_i for links 2 and 3 (Fig. 71) and the static friction at all joints. To accomplish this, force measurements were taken for eight arm configurations. Figure 72 schematically illustrates the eight test configurations. In each configuration i, the minimum applied force $F^i_{a,min}$ and maximum applied force $F^i_{a,max}$ to allow the free joint to just begin to move were measured. Each measurement was taken five times and Table X lists the average value and standard deviation for each of these measurements. The static friction for configuration i, F^i_f, can be approximated by

$$F^i_f = \frac{F^i_{a,max} - F^i_{a,min}}{2}$$

and the support force F^i_g needed to counteract gravity is the average applied force

$$F^i_g = \frac{F^i_{a,max} + F^i_{a,min}}{2}.$$

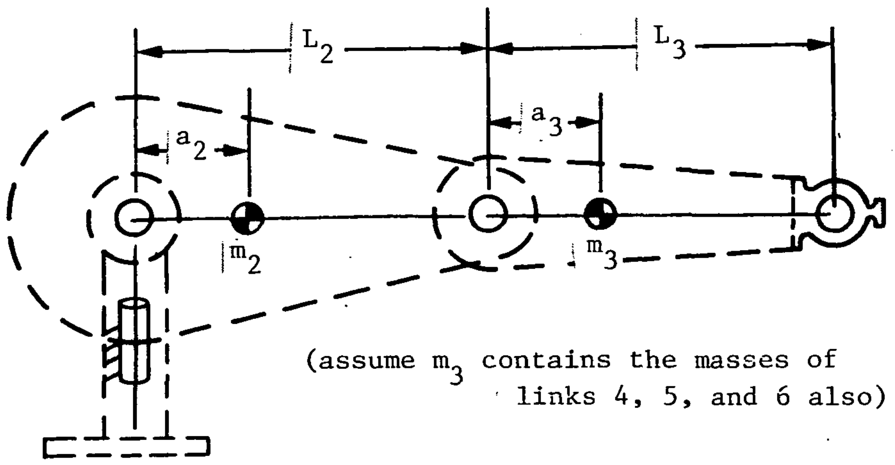

L_2
L_3
a_2
a_3
m_2
m_3
(assume m_3 contains the masses of links 4, 5, and 6 also)

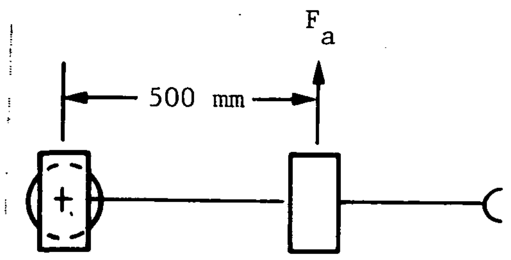
Figure 71. - PUMA mass/centroid parameters.

TABLE X. - MEASURED FRICTION AND GRAVITY FORCES

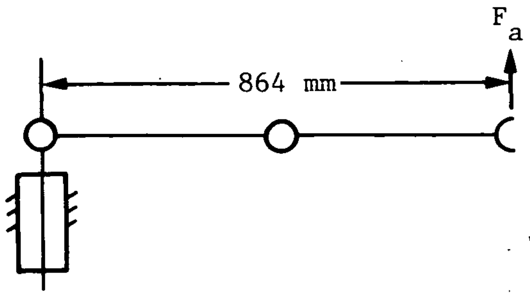
i	$F^i_{a,min}/\sigma$ *	$F^i_{a,max}/\sigma$	F^i_g	F^i_f
a	-35.6/0.49	39.4/0.55	-	37.5
b	51.2/089	66.7/0.53	58.9	7.78
c	13.4/0.12	26.6/0.47	20.0	6.58
d	80.1/0.67	112.5/0.84	96.3	16.2
e	38.7/1.07	64.9/0.84	51.8	13.1
f	11.8/0.71	23.1/0.31	17.5	5.65
g	10.0/0.22	24.7/0.18	17.4	7.34
h	12.0/0.09	22.6/0.11	17.3	5.32

* σ - standard deviation; all values given in newtons.

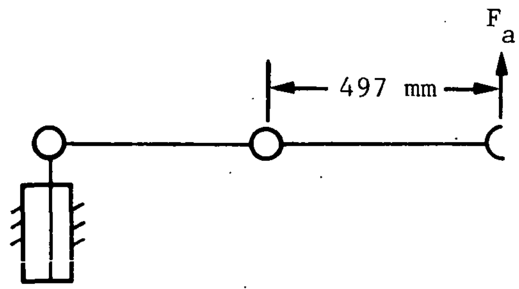
Based on the results listed in Table X and the effective moment arms for the different configurations shown in Figure 72, the joint friction forces can be evaluated. Table XI lists the resulting joint friction torques. Note that the configurations b, d, and e all give values for the static friction at joint 2. The values from configurations b and d are 6.72 and 6.82 Nm, but the measured value from configuration e is 8.82 Nm which is inconsistent with the other results. Unfortunately, the test could not be readily repeated, so it is unclear whether this indicates a true anomaly or that an error was made in recording the arm configuration. The latter explanation seems unlikely because the gravity force in this configuration agrees closely with the results for configuration b, as discussed later.



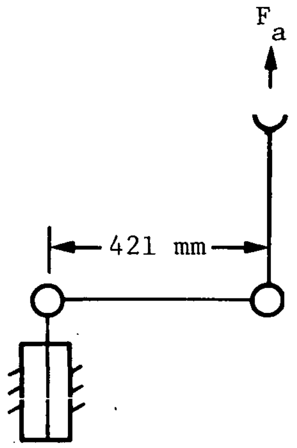
(a) Joint 1 free.



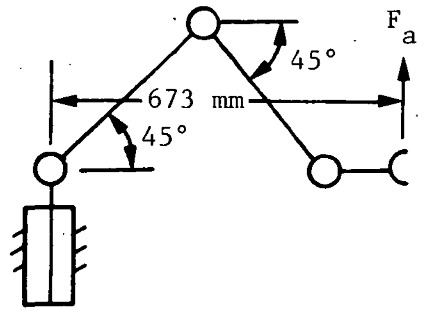
(b) Joint 2 free.



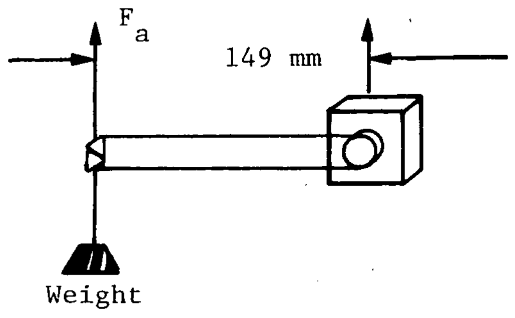
(c) Joint 3 free.



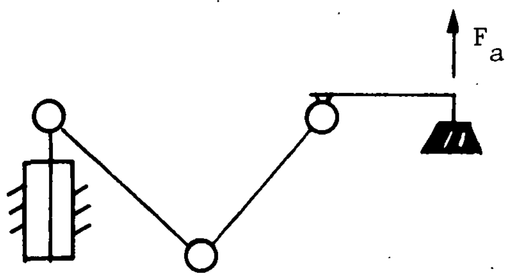
(d) Joint 2 free.



(e) Joint 2 free.



(f) Joint 4 free.
(g) Joint 6 free.



(h) Joint 5 free.

Figure 72. - Static load measurement configurations.

TABLE XI. - JOINT STATIC FRICTION TORQUES

Joint	Static Friction, Nm
1	18.8
2	6.8
3	3.3
4	0.84
5	0.84
6	1.09

Analysis of the results in this and the preceding section provide the following conclusions and suggested modifications of the testing procedure:

1) The standard deviation for each set of measurements is quite small (approximately 1%), providing good confidence in the resulting values;

2) The F_g^i values for measurement configurations f, g, and h agree closely, as they should, because the supported weight is the same in each of these cases;

3) The directional nature of the friction forces must be evaluated. As seen from the results from joint 1 of the PUMA and from the planar arm, the friction coefficient for one direction of motion can significantly differ from the value for the other direction;

4) The dry friction when the joint is just barely moving should be evaluated in addition to the static value determined by these tests;

5) The friction forces from the input-driven system should be compared with the values obtained by the technique presented here in which the output is driven.

The force measurements from configurations b, c, d, and e also provide values for the mass and centroid locations.

The discussion in the adaptive control subsection of this report showed that only certain combinations of these terms can be identified, i.e., response measurements cannot differentiate between some of the terms. This also means that these observable mass combinations determine the dynamic properties and the individual components need not be evaluated explicitly. The disassembled manipulator must be measured or engineering approximations must be used to explicitly evaluate the individual terms. In the measurement cases considered here, the terms that can be identified are $D_{15} = m_3a_3$ and $D_{14} = m_2a_2 + m_3L_2$.

For example, configuration c provides the result (Fig. 71 and 72)

$$m_3a_3g = F_ad = (20.0N)(0.497m)$$

where $g = 9.8\ m/s^2$ is the acceleration caused by gravity. This leads to

$$D'_{15} = m_3a_3 = 1.01\ kg\text{-}m.$$

Similarly, configuration d provides the value for D'_{14} as

$$g[m_2a_2 + m_3L_2] = F_ad = (96.3N)(0.421)m$$

$$D'_{14} = m_2a_2 + m_3L_2 = 4.14\ kg\text{-}m.$$

These values can be validated by comparing them with the measurements from configurations b and e. For configuration b, the values given above yield

$$g[m_2a_2 + m_3(L_2 + a_3)] = g(D'_{14} + D'_{15}) = 50.5\ Nm,$$

which is within 1% of the measured value

$$F_ad = (58.9N)(0.864m) = 50.9\ Nm.$$

Also, for configuration e, the identified mass properties produce

$$g \cos 45°[m_2a_2 + m_3(L_2 + a_2)] = 0.7071\ g\ (D'_{14} = D'_{15}) = 35.7\ Nm,$$

which differs by about 2% from the measured value

$$F_ad = (51.8N)(0.673m) = 34.9\ Nm.$$

In addition to the static force measurements for determining the system parameters, dynamic response was also measured. These tests consisted of running certain trajectories with a single joint moving at a time and recording the position and driving current profiles during the motion.

The data from the dynamic response runs were transferred to Martin Marietta. The intent of the effort was to first plot position, velocity and acceleration profiles of the single-joint PUMA runs. The PUMA would then be modeled using ROBSIM and response simulations would be run using the same currents as used to run the actual PUMA arm. After this, positions, velocities, and accelerations of the simulated motions would be plotted and compared with the joint motions of the actual PUMA. However, when the motion profiles for the actual hardware joints were plotted, a lot of noise was present. Figures 73, 74 and 75 show position, velocity and acceleration plots for joint 3 of the PUMA and the noise present in the signals. At this time of this writing the source of the noise has not been determined and further testing is necessary for this evaluation.

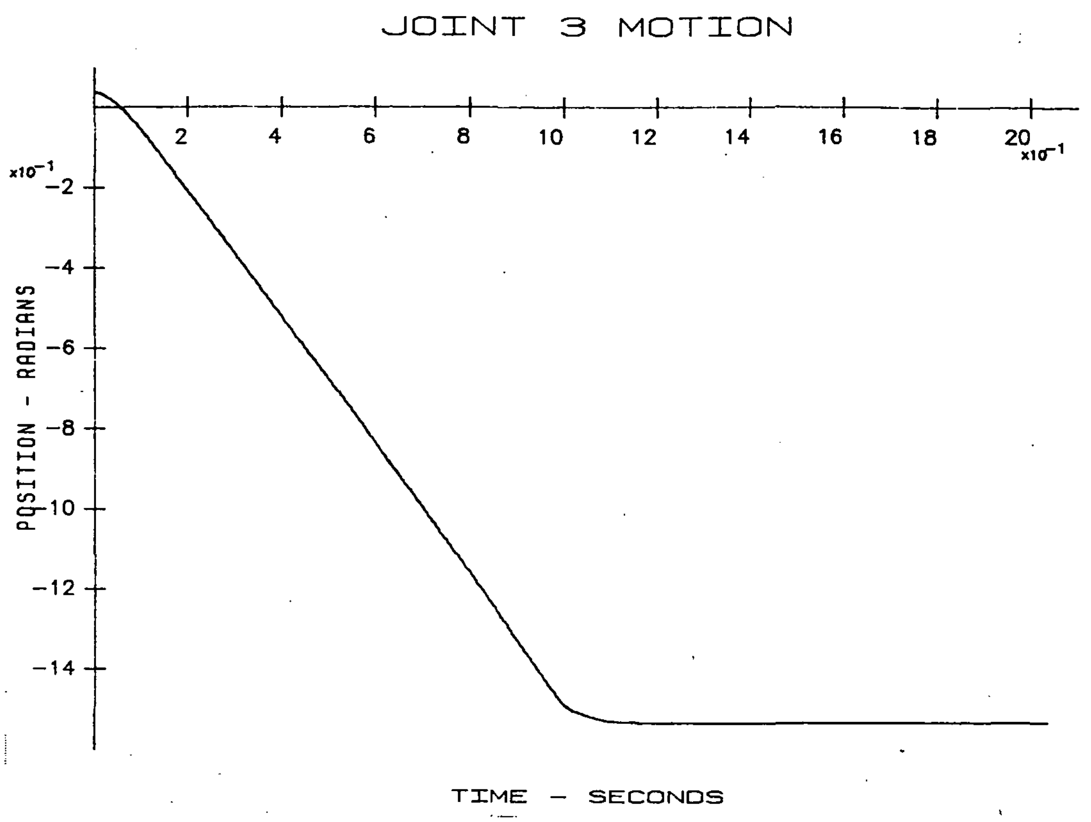


Figure 73. - PUMA joint 3 position profile.

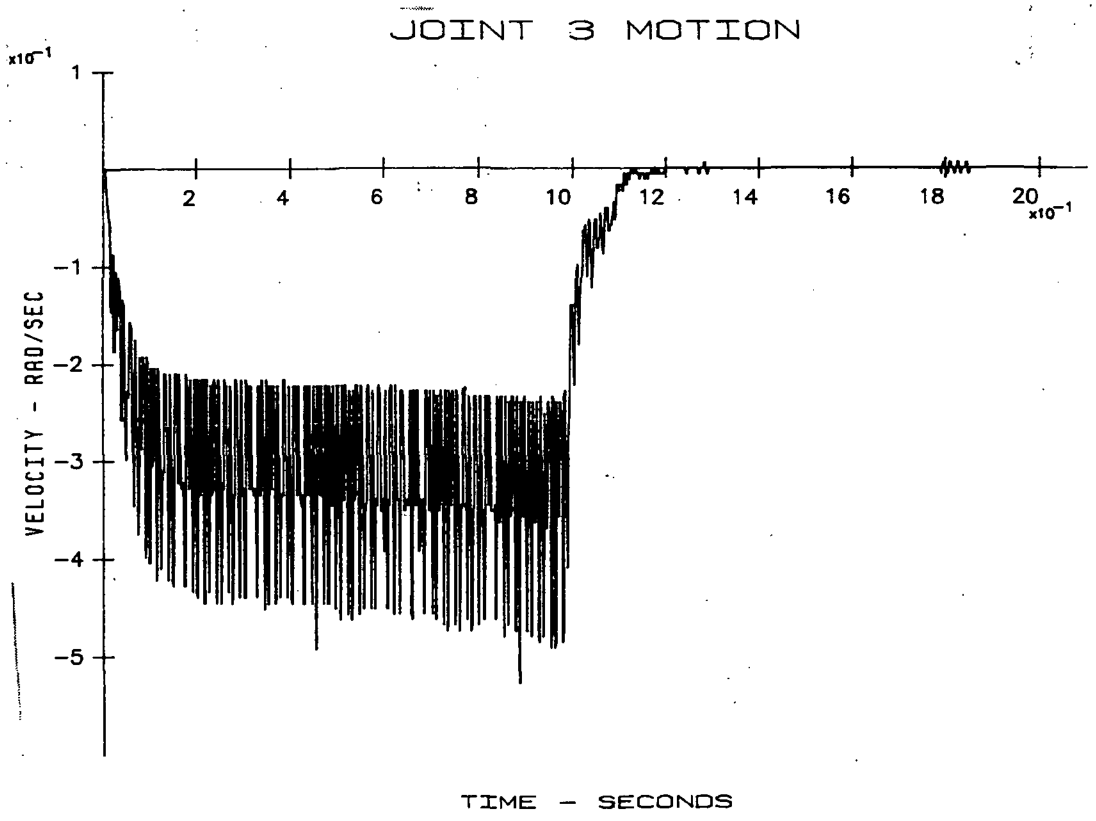


Figure 74. - PUMA joint 3 velocity profile.

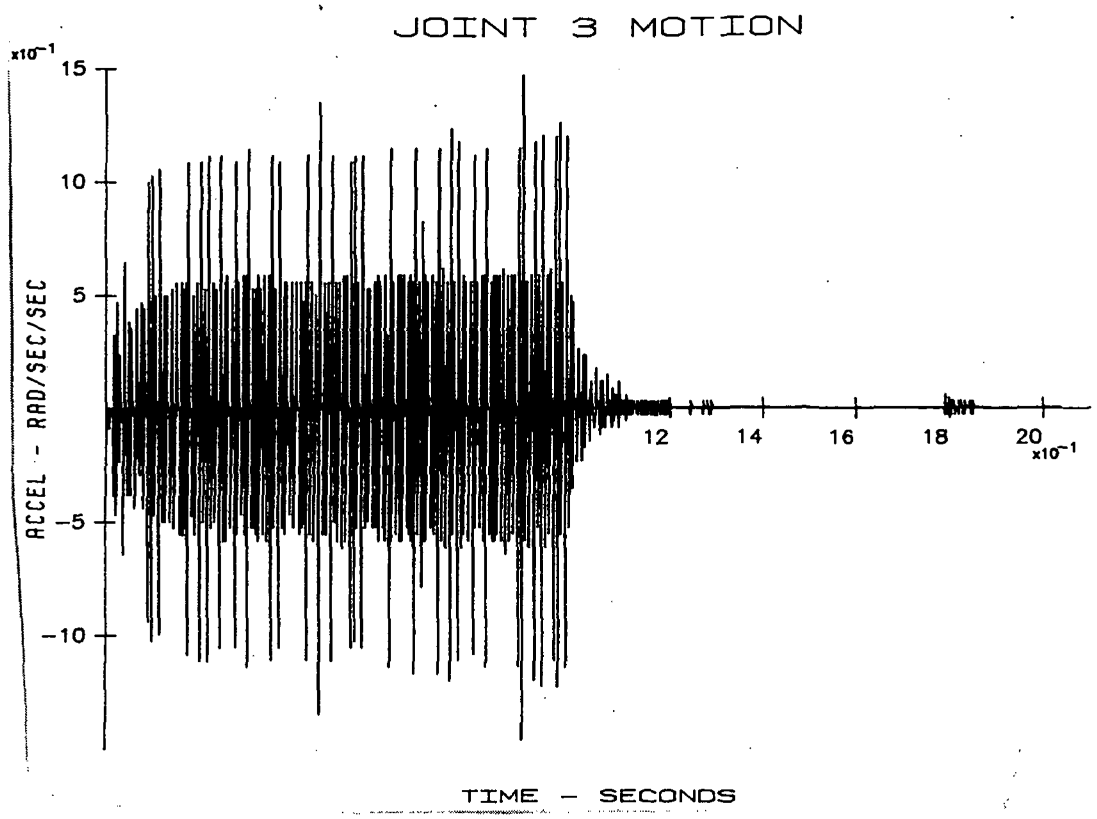


Figure 75. - PUMA joint 3 acceleration profile.

CONCLUDING REMARKS

This report has documented the work done in Phases II and III of contract NAS1-16759, Evaluation of Automated Decisionmaking Methodologies and Development of an Integrated Robotic System Simulation. The tasks defined for Phase II (tasks 9 thru 13) and Phase III (14 thru 20) have significantly increased the capabilities of the robotic simulation (ROBSIM) package and the technologies associated with the automation and operation of advanced manipulator systems listed:

1) System definition - The implementation of this module allows a user to define a manipulator system consisting of multiple arms, load objects and an environment. This addressed task 12 of the contract;

2) Analysis capabilities - This has added the capabilities of kinematic analysis, requirements analysis, and response simulation for investigating manipulator motion. This addressesed tasks 14, 15, and 16;

3) Postprocessing - This module allows the user to better evaluate system analysis results through the graphic replay of simulated motion or manipulator parameter plots;

4) Control - Various control methods were investigated and those implemented in ROBSIM include manual force/torque and active compliance control. This addressed tasks 13, 17 and 19 of the contract;

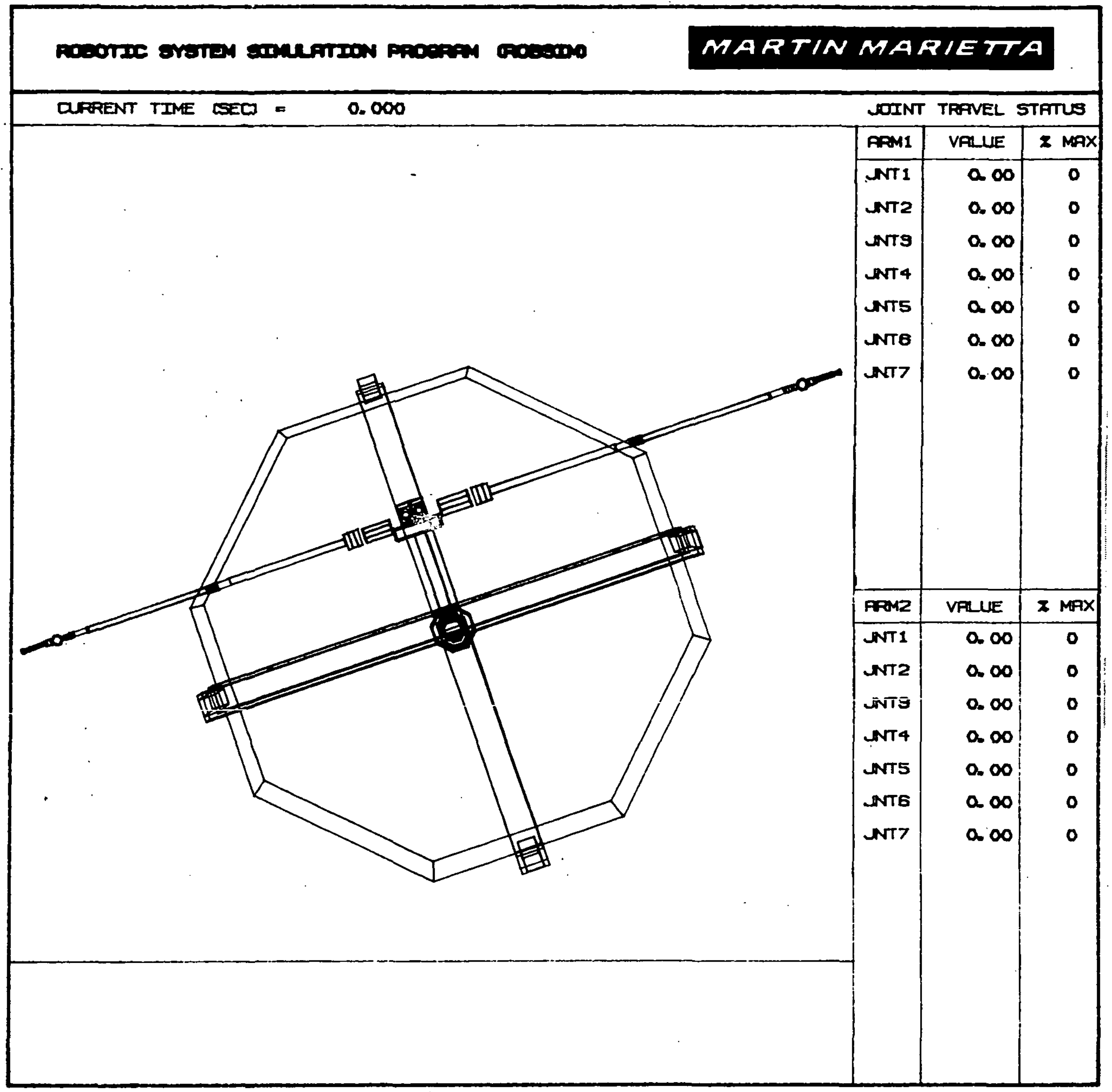
5) Trajectory planning - Three methods of obstacle avoidance were implemented and evaluated and research was conducted in the field of trajectory planning. This work addressed tasks 9 and 18;

6) Image processing - The module implemented video simulation and edge detection and addressed tasks 10 and 11;

7) Simulation validation - The software simulation was validated in conjunction with the Automation Technology Branch of NASA-LRC. This work addresses task 20 of the contract.

Target areas for future enhancement of the ROBSIM capabilities include greater ease of user interaction with the program, expanded manipulator modeling abilities, and the addition of system compliance in the model.

Expansion of the ROBSIM software could now be directed toward support of teleoperator and robotic systems capable of such remote space operations as the remote orbital servicing system (ROSS) concept. The simulation could support dual manipulator arms attached to a carrier spacecraft in orbit. Forces, moments and motions typical of a dual-arm robot/host vehicle system of this type would be included in the kinematic and dynamic solutions. It is anticipated that the models for the manipulator motors, the sensors and end-effector will warrant the development of new software modules and laboratory validation experiments. Figure 76 is a sample of a preliminary ROSS simulation graphics display.



ROBOTIC SYSTEM SIMULATION PROGRAM (ROBSIM)

CURRENT TIME (SEC) = 0.000

JOINT TRAVEL STATUS

ARM1	VALUE	% MAX
JNT1	0.00	0
JNT2	0.00	0
JNT3	0.00	0
JNT4	0.00	0
JNT5	0.00	0
JNT6	0.00	0
JNT7	0.00	0

ARM2	VALUE	% MAX
JNT1	0.00	0
JNT2	0.00	0
JNT3	0.00	0
JNT4	0.00	0
JNT5	0.00	0
JNT6	0.00	0
JNT7	0.00	0

Figure 76. - Preliminary ROSS simulation graphics display.

An investigation of concepts for a system that allows manual designation of objects within a visual (television) scene and that performs high-accuracy position measurements could be conducted. A description of one or more systems that provides this capability for the ROSS configuration would be developed.

Continuing work in the trajectory planning field would couple the man/machine interface with the software-simulation of remote systems. Assessment of the problem of path trajectory singularity avoidance and various approaches to autonomous task planning should be researched. A task-oriented command module that incorporates task primitive commands into the simulation user interface would be documented and developed. Executing a task command will initiate a sequence of appropriate sensor, actuator and state commands. Techniques employing rule-based systems and theorem provers may solve the problems associated with task scheduling to achieve a specific mission goal.

Validation of the existing software simulation modules will continue, and the models for actuators, joints and links should be extended to fit a more generalized set of hardware robotic systems. To effect a better representation of the robotic hardware defined in the system development phase of the program, an interface to allow transfer and conversion of CAD/CAM (computer-aided design and manufacturing) or geometric modeling data to the ROBSIM format should be designed. The establishment of a compatible exchange of product definition between CAD/CAM and ROBSIM would enhance the general applicability of ROBSIM-generated databases and serve to lessen duplication of efforts in modeling hardware component systems. The current method for definition of robotic systems within ROBSIM proceeds by defining primitive robotic hardware geometry and provides only rough sketches of the actual hardware properties. Figure 77 depicts a CAD/CAM robot arm that was converted from a standardized IGES (initial graphics exchange specification) formatted file for display on a vector graphics terminal.

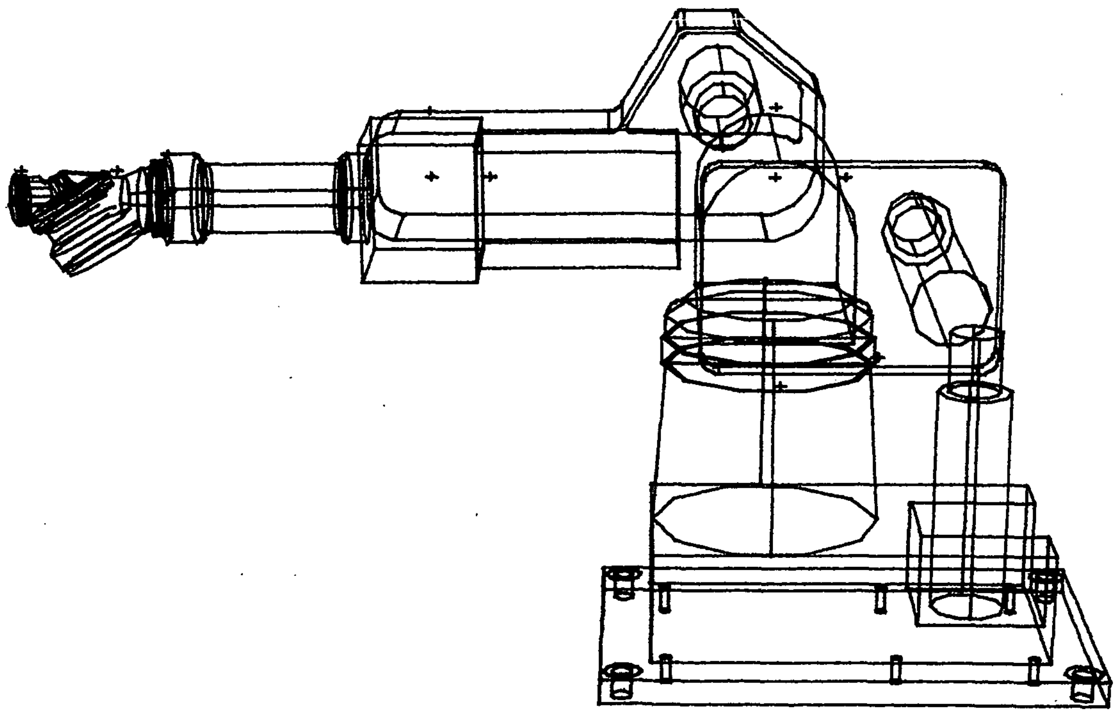


Figure 77. - Display of robot arm converted from a CAD/CAM database.

Current processing capabilities in the ROBSIM program may be carried out on virtually any modern digital computing machine of the supermini or higher class. However, with the exception of the newest and fastest machines, the necessary speed for real-time simulation processing will not be achieved by a single machine. Specialized machines with multiple processor architectures will be necessary to accomplish this goal. Areas such as mechanical configuration optimization, accuracy analysis, control requirements, stability, and work environment studies require simulation but not necessarily in real time. Several significant long-range requirements point to the need for a real-time simulation capability, and research of the following available systems is recommended:

1) Realistic graphics - When the simulation system is coupled with a graphics display of the computer solutions in real time, the results are much more realistic;

2) Surrogate hardware - Properly simulated hardware may be used to replace hardware subsystems that have failed or are not yet available. The remainder of the hardware may be used in the system as intended. System problems may be isolated by substituting software modules for suspected failing hardware modules. System integration may be approached more gradually by bringing hardware on line one module at a time or as the hardware is available;

3) Real time testbed for algorithm development - Control and estimation algorithms may be evaluated and optimized using the true or emulated control computer without the use of scaling within the algorithms, and with no potential damage to the existing and manipulator assembly;

4) *Real-time testbed for path planning and man/machine interface - Safe,* real-time response to decisions and trajectory decisions as dictated by sensor output and command input is very important for a smooth flow of the man/machine interface within the simulation and to achieve a diversified approach to task-oriented plans.

REFERENCES

Ballard, D. H.; and Brown, C. M.: Computer Vision. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982.

Barker, L. Keith: Vector Algebra Approach to Extract Denavit-Hartenberg Parameters of Assembled Robot Arms. NASA Technical Paper 2191, August 1983.

Binford, T. O.: Inferring Surfaces from Images. Artificial Intelligence, vol. 17, 1981, pp. 205-244.

Brady, M: Computational Approaches to Image Understanding. Computing Surveys, vol 14 (1), 1982, pp. 3-71.

Chace, M. A.; Calahan, D. A.; Orlandea, N.; and Smith, D.: Formulation and Numerical Methods in the Computer Evaluation of Mechanical Dynamic Systems. Third World Congress for the Theory of Machines and Mechanisms, Kupari, Yugoslavia, 1971, pp. 61-100.

Dubowsky, S.; and DesForges, D. T.: The Application of Model-Referenced Adaptive Control to Robotic Manipulators. Joint Automatic Control Conference, 1979.

Duffy, J,; and Crane, C.: A Displacement Analysis of the General Spatial 7-link, 7R Mechanism. Mechanism and Machine Theory, vol. 15, no. 3, 1980, pp. 153-169.

Duffy, J,: Analysis of Mechanisms and Robot Manipulators. Wiley, New York, NY, 1981.

Duffy, J,: Analysis of Robot Arms. DOE Contract AC05-79ER-10013, Center of Intelligent Machines and Robotics, University of Florida, Gainesville, FL, 1979.

Franklin, G. F.; and Powell, J. D.: Digital Control of Dynamic Systems. Addison-Wesley Publishing Company, Menlo Park, CA, 1981.

Fukunaga, Keinosuke: Introduction to Statistical Pattern Recognition. Academic Press, San Francisco, 1972, p. 294.

Goodwin, Graham C.; and Sin, K. S.: Adaptive Filtering Prediction and Control. Manuscript, Department of Electrical and Computer Engineering, University of Newcastle, New South Wales, 2308, Australia, 1982.

Hanson, A. R.; and Riseman, E. M.: Processing Cones: A Computational Structure for Image Analysis. Structured Computer Vision, ed. Tanimoto, S.; and Klinger, A.; Academic Press, New York, NY, 1980.

Hemani, J; and Camana, P. C.: Nonlinear Feedback in Simple Locomotion Systems. IEEE Transactions on Automatic Control, December 1976.

Hildreth, E. C.: Edge Detection for Computer Vision Systems. Mechanical Engineering, August 1982, pp. 48-53.

Hollerback, J. M.: A Recursive Lagrangian Formulation of Manipulator Dynamics and a Comparative Study of Dynamics Formulation Complexity. IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-10, no. 11, 1980, pp. 730-736.

Horowitz, R.; and Tomizuka, M.: Discrete-Time Model Reference Adaptive Control of Mechanical Manipulators. ASME Computer Engineering Conference, August, 1982.

Hueckel, M. F.: An operator Which Locates Edges in Digitized Pictures. Journal of the ACM, vol. 18, 1971, pp. 113-125.

Hummel, R. A.: Edge Detection Using Basic Functions. Technical Report TR-56, University of Maryland, Computer Science Center, August 1977.

Koivo, A. J.; and Guo, T. H.: Control of Robotic Manipulator with Adaptive Controller. Conference on Decision and Control, December 1981.

Kreisselmeier, G.: On Adaptive State Regulation. IEEE Transactions, vol. AC-27, no. 1, February 1982.

Lee, C. S. G.: Robot Arm Kinematics, Dynamics, and Control. Computer Magazine, December 1982, pp. 62-80.

Lee, C. S. G.; and Chung, M. J.: An Adaptive Control Strategy for Mechanical Manipulators. Proceedings, 21st Conference on Decision and Control, 1982.

Lee, George: University of Michigan Department of Electrical and Computer Engineering, Ann Arbor, MI. Personal Correspondence.

Leinenger, G. G.: Self-Tuning Adaptive Control of Manipulators. Symposium on Advanced Software in Robotics, Liege, Belgium, 1983.

Lowrie, J. W.; Formelia, A. J.; Haley, D. C.; Gremban, K. D.; Van Baalen, J.; and Walsh, R. W.: Evaluation of automated Decisionmaking Methodologies and Development for an Integrated Robotic System Simulation. NASA Contractor Report 165975, 1982.

Luh, J. Y. S.; Walker, M. W.; and Paul, R. P. C.: Resolved-Acceleration Control of Mechanical Manipulators. IEEE Trans. vol. AC-25, no. 3, June 1980.

Luh, J. Y. S.; Walker, M. W.; and Paul, R. P. C.: Online Computational Scheme for Mechanical Manipulators. J. Dynamic System. Measurement. Contr., Transactions ASME, vol. 102, June 1980.

MacVicar-Whelan, P. J.; and Binford, T. O.: Line Finding with Subpixel Precision. Techniques and Applications of Image Understanding, ed. Pearson, J. J.; SPIE vol. 281, 1981, pp. 211-216.

Marr, D.; and Hildreth, E. C.: Theory of Edge Detection. Proceedings of the Royal Society of London, Series B, 207, 1980, pp. 187-217.

Mason, M. T.: Compliance and Force Control for Computer-Controlled Manipulators. IEEE Transactions, vol. SMC-11, no. 6, June 1981.

Morgenthaler, D. G.: A New Hybrid Edge Detector. Computer Graphics and Image Processing, vol. 16, 1981, pp. 166-176.

Nevins, J. L.; and Whitney, D. E.: Computer-Controlled Assembly. Scientific America, 1978.

Nishihara, K.; and Larson, W.: Towards a Real-Time Implementation of the Marr and Poggio Stereo Matcher. Techniques and Applications of Image Understanding, ed. Pearson, J. J.; SPIE vol. 281, 1981, pp. 299-305.

O'Gorman, F.: Edge Detection Using Walsh Functions. Artificial Intelligence, vol. 10, 1978, pp. 215-223.

Okada, T.: Computer Control of Multijointed Finger System for Precise Object Handling. IEEE Sys. Man. Cyber, vol. SMC-12, no. 3, 1982.

Orin, D. E.; McGhee, R. B,; Vukobratovic, M.; and Hartoch, G.: Kinematic and Kinetic Analysis of Open-Chair linkages Utilizing Newton-Euler Methods. Mathematical Biosciences, vol. 43, 1979, pp. 107-130.

Paul, R. P.: Robot Manipulators: Mathematics, Programming, and Control. MIT Press, Cambridge, Massachusetts, 1981a.

Paul, R. P.: Kinematic Control Equations for Manipulators. IEEE Transactions, vol. SMC-11, 1981b.

Paul, R. P.; MaRong; Hong Zhang: The Dynamics of the PUMA Manipulator. American Control Conference Proceedings, 1983, pp. 491-496.

Peli, T.; and Malah, D.: A Study of Edge Detection Algorithms. Computer Graphics and Image Processing, vol. 20, 1982, pp. 1-21.

Prewitt, J. M. S.: Object Enhancement and Extraction. Picture Processing and Psychopictories, ed. Lipkin, B. S.; and Rosenfeld, A.; Academic Press, New York, NY, 1970.

Raibert, M. H.; and Craig, J. J.: Hybrid Position/Force Control of Manipulators. ASME Transactions, Journal of Dynamic Systems, Measurement, and Control, vol. 102, pp. 126-133.

Rosenfeld, A.; and Kak, A. C.: Digital Picture Processing. Second Edition, Academic Press, New York, NY, 1982.

Rosenfeld, A.; and Thurston, M.: Edge and Curve Detection for Visual Scene Analysis. IEEE Transactions, vol. C-20(5), 1972, pp. 562-569.

Salisbury, J. K.: Active Stiffness Control of Manipulator in Cartesian Coordinates. Proceedings of the 19th IEEE Conference of Decision and Control, 1980, pp. 95-100.

Shanmugan, K. S.; Dickey, F. M.; and Green, J. A.: An Optimal Frequency Domain Filter for Edge Detection in Digital Pictures. IEEE Transactions, vol. PAMI-1, 1979, pp. 37-49.

Shimano, b. E.; and Roth, B.: On Force Sensing Information and Its Use in Controlling Manipulators. Proceedings of the 9th International Symposium on Industrial Robots, Washington, D.C., 1979. pp. 119-126.

Shirai, Y.: Recognition of Real-World Objects Using Edge Cues. Computer Vision Systems, ed. Hanson, A. R.; and Riseman, E. M.; Academic Press, New York, NY, 1978.

Sugimoto, K.; Duffy, J,; and Hunt, K. H.: Special Configurations of Spatial Mechanisms and Robots. Mechanism and Machine Theory, vol. 17, no. 2, 1982, pp. 119-132.

Thomas, M.; and Tesar, D.: Dynamic Modeling of Serial Manipulator Arms. ASME Transactions, Journal of Dynamic Systems, Measurement, and Control, vol. 104, 1982, pp. 218-228.

Walker, M. W.; and Orin, D. E.: Efficient Dynamic Computer Simulation of Robotic Mechanisms. ASME Transactions, Journal of Dynamic Systems, Measurements, and Control, vol. 104, 1982, pp. 205-211.

Whitney, D. E.: Resolved Motion Rate Control of Manipulators and Human Prosthesis. IEEE Transactions, vol. MMS-10, no. 2, 1969, pp. 47-53.

Wu, C. H.; and Paul, R. P.: Resolved Motion Force Control of Robot Manipulator. IEEE Transactions, vol. SMC-12, no. 3, 1982, pp. 266-275.

1. Report No. NASA CR-172401	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle Evaluation of Automated Decisionmaking Methodologies and Development of an Integrated Robotic System Simulation		5. Report Date September 1984 6. Performing Organization Code
7. Author(s) D. C. Haley, B. J. Almand, M. M. Thomas, L. D. Krauze, K. D. Gremban, J. C. Sanborn, J. H. Kelly, T. M. Depkovich		8. Performing Organization Report No. MCR-84-549
9. Performing Organization Name and Address Martin Marietta Aerospace Denver Aerospace P.O. Box 179 Denver, CO 80201		10. Work Unit No. 11. Contract or Grant No. NAS1-16759
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546		13. Type of Report and Period Covered Contractor Report 14. Sponsoring Agency Code

15. Supplementary Notes

Langley Technical Monitor: Jack Pennington
Report Consists of Three Volumes: Study Results, Appendix A—User's Guide, Appendix B—Programmer's Guide

16. Abstract

This report covers the work performed on tasks 9—20 of contract NAS1-16759. The work done under this contract implemented a generic computer simulation for manipulator systems (ROBSIM) and developed the specific technologies necessary to increase the role of automation in various missions. The specific items developed are:

1) Capability for definition of a manipulator system consisting of multiple arms, load objects, and an environment;
2) Capability for kinematic analysis, requirements analysis, and response simulation of manipulator motion;
3) Postprocessing options such as graphic replay of simulated motion and manipulator parameter plotting;
4) Investigation and simulation of various control methods including manual force/torque and active compliance control;
5) Evaluation and implementation of three obstacle avoidance methods;
6) Video simulation and edge detection;
7) Software simulation validation.

The tasks listed above are documented in the study results volume of this report. Appendix A is the user's guide and includes examples of program runs and outputs as well as instructions for program use. Appendix B, the programmer's guide, defines the program structure and includes a VCLR and short explanation for each subroutine.

17. Key Words (Suggested by Author(s)) ROBSIM		18. Distribution Statement Unclassified - Unlimited Subject Category 63	
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 170	22. Price

N-305